НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ
(повна назва інституту/факультету)

кафедра БІОМЕДИЧНОЇ КІБЕРНЕТИКИ
(повна назва кафедри)

«На правах рукопису»
УДК 004.021

«**До захисту допущено**»
Завідувач кафедри БМК

_____ Є.А. НАстенко
(підпис) (ініціали, прізвище

“___”_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 122 «Комп’ютерні науки та інформаційні технології»
(код і назва)

на тему: «Створення та порівняльний аналіз кластеризаційних алгоритмів розпізнавання статистичних залежностей»

Виконав (-ла): студент (-ка) **VI** курсу, групи БС-61м
(шифр групи)

УМАНЕЦЬ ВІТАЛІЙ СЕРГІЙОВИЧ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник *доц. каф. БМК., к.т.н. Павлов В.А.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з розділів МД *зав. каф БМК, д.б.н., к.т.н., с.н.с., Настенко Є. А.*
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент *доц. каф. БМІ, к.т.н., Зубчук В.І* _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент *доц. каф ФВ, к.п.н., Бойко Г.Л.* _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____
(підпис)

Київ – 2018 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут (факультет) БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ
(повна назва)

Кафедра БІОМЕДИЧНОЇ КІБЕРНЕТИКИ
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-науковою програмою
спеціальність 122 «Комп'ютерні науки та інформаційні технології»
(спеціалізація) (Інформаційні технології в біології та медицині)
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри БМК
_____ _Є.А. Настенко_
(підпис) (ініціали, прізвище)
«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

УМАНЦЮ ВІТАЛІЮ СЕРГІЙОВИЧУ
(прізвище, ім'я, по батькові)

1. Тема дисертації «Створення та порівняльний аналіз кластеризаційних алгоритмів розпізнавання статистичних залежностей»
науковий керівник дисертації
Павлов Володимир Анатолійович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «_29_» березня_ 2018 р. №_1041-с

2. Термін подання студентом дисертації ***11-12 травня 2018 року***
3. Об'єкт дослідження Технології Data Mining

4. Предмет дослідження Кластерний аналіз

5. Перелік завдань, які потрібно розробити
Провести аналіз сучасних алгоритмів кластеризації, розробити алгоритм нечітких k-середніх з обмеженою масою робочої області, спроектувати та

реалізувати програмний засіб для побудови розбиття множини об'єктів на кластери за допомогою розробленого алгоритму, оцінити ефективність алгоритму на практичній задачі

6. Орієнтовний перелік графічного (ілюстративного) матеріалу

Таблиці, які демонструють роботу алгоритму, зображення, які характеризують результати роботи програми.

7. Орієнтовний перелік публікацій студентів у просторі параметрів залежності

1. Аналіз станів системи кровообігу артеріальний тиск-пульс / К: Вісник університету "Україна", Серія "Інформатика, обчислювальна техніка та кібернетика" 2018. № 1(21), 13 с. 2. Метод нечітких k-середніх з обмеженою масою робочої області формування кластерів довільної форми // Біомедична інженерія і технологія.
3. Модифицированный алгоритм С-средних для функционально связных кластеров // Теорія і практика наукових знань (частина IV):матеріали II Міжнародної науково-практичної конференції м. Київ, 28-29 грудня 2017 р – Київ.: МЦНД, 2017. С. 48-49

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Магістерської дисертації	Настенко Є. А., зав. Каф. БМК, д.б.н., к.т.н., с.н.с.,	19.03.18	03.05.18

9. Дата видачі завдання ***19 березня 2018 р.***

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Отримати завдання на МД	19 березня 2018р..	
2	Огляд літератури	30 березня 2018р.	
3	Побудова структури програмної системи	10 березня 2018р.	
4	Проведення експериментів	10 квітня 2018р.	
5	Написання МД	1 травня 2018р.	
6	Предзахист МД та допуск до захисту дисертації	3 травня 2018р..	
7	Подання МД рецензенту. Отримання рецензії.	4-7 травня 2018р.	
8	Подання в електронному вигляді МД та анотації до неї на сайт кафедри.	11-12 травня 2018р.	
9	Подання пакету документів по МД до захисту в ЕК	11-12 травня 2018р.	
10	Захист МД в ЕК	18-19 травня 2018р..	

Студент _____ Уманець В.С.
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Павлов В.А.
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено науковго керівника магістерської дисертації.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

JVM – Java Vitual Machine

ПЗ – програмне забезпечення

ПП – програмний продукт

DFD – Data Flow Diagram

ВСТУП

Актуальність теми. Інтелектуальний аналіз даних пов'язаний із пошуком закономірностей, що дозволять отримати нову інформацію про досліджувані дані. Останнім часом інтерес до засобів та методів аналізу даних підвисився у зв'язку з розвитком технологій для зберігання та обробки великих масивів даних. В той час як традиційні статистичні методи аналізу даних займаються здебільшого оцінкою параметрів для вже виведених закономірностей, методи інтелектуального аналізу даних дозволяють виявити нові, скриті закономірності в досліджуваних даних.

Задача визначення функціонального зв'язку між біофізичними показниками є складовою частиною актуального завдання пошуку оптимальних впливів на біологічний об'єкт і не вирішена в повній мірі на сьогоднішній день. При цьому найбільш цікавими є результати, що адекватно представляють розбиття простору на області (кластери) які відносяться до різних функціональних співвідношень, що зв'язують біофізичні показники, що розглядуються в даній області. Такі кластери логічно називати функціональними, а їх форма в загальному випадку може бути довільною. Для адекватного поділу вихідної сукупності на такі однорідні групи необхідне застосування нових інформаційних технологій.

Мета роботи – розробити алгоритм кластерного аналізу, що дозволяє формувати кластери довільної форми та оцінювати динаміку змін характеристик біологічних об’єктів.

У відповідності до мети сформовано наступні **задачі**:

- Провести аналіз сучасних алгоритмів кластеризації.
- Розробити алгоритм нечітких k-середніх з обмеженою масою робочої області.

- Спроектувати та реалізувати програмний засіб для побудови розбиття множини об'єктів на кластери за допомогою розробленого алгоритму.
- Оцінити ефективність алгоритму на практичній задачі.

Об’єкт дослідження – технології Data Mining.

Предмет дослідження – кластерний аналіз.

Для досягнення мети дослідження та реалізації програмної системи використано IntelliJ IDEA з використанням мов програмування Java та Scala.

Реалізація результатів роботи.

Робота виконана на замовлення Кафедри фізичного виховання. Одержані результати дослідження впроваджені в діяльність Кафедри фізичного виховання. (акт впровадження від 27.04.18).

Апробація результатів роботи.

Основні положення та результати магістерської дисертації були викладені в наступних роботах:

1) Аналіз станів системи кровообігу студентів у просторі параметрів залежності артеріальний тиск-пульс // Вісник університету "Україна", Серія "Інформатика, обчислювальна техніка та кібернетика" 2018. № 1(21), 13 с.

2) Метод нечітких k-середніх з обмеженою масою робочої області формування кластерів довільної форми // Біомедична інженерія і технологія, 7 с.

3) Модифицированный алгоритм С-средних для функционально связных кластеров // Теорія і практика накових знань (частина IV): матеріали II Міжнародної науково-практичної конференції м. Київ, 28-29 грудня 2017 року. – Київ.: МЦНД, 2017. С. 48-49.

Структура дисертації

Дисертація побудована за класичним типом та викладена на 110 сторінках машинописного тексту. Складається з вступу, 5 розділів,

висновків, списку використаних літературних джерел, який містить 49 найменувань, 29 – на кирилиці, 20 – на латиниці. У роботі представлено 68 рисунків і 5 таблиць.

РОЗДІЛ 1
ЗАГАЛЬНІ ВІДОМОСТІ З ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Data Mining

Data Mining, в області інформатики, процес відкриття цікавих та корисних патернів та відношень у великих об’ємах даних [1]. Data Mining поєднує інструменти зі статистики та штучного інтелекту (наприклад, нейронних мереж та машинного навчання) з управлінням базами даних для аналізу великих цифрових колекцій, відомих як набори даних. Видобуток даних широко використовується в бізнесі (страхування, банківська справа, роздрібна торгівля), наукових дослідженнях (астрономія, медицина) та державної безпеки (виявлення злочинців та терористів).

Основними етапами отримання знань з даних є:

- Очищення даних від шумів.
- Об'єднання даних, щ обули отримані з різних джерел.
- Відбір тільки тих даних, що необхідні для вирішення поставленої задачі.
- Представлення даних у зручній для аналізу формі.
- Безпосередньо використання методів пошуку шаблонів у даних.
- Аналіз отриманих шаблонів.
- Візуалізація отриманих знань.

На рис. 1.1 послідовність виконання описаних вище етапів зображено у більш згорнутому вигляді.

Отримані у результаті аналізу шаблони можуть бути як описувальні, так і прогнозуючі. Останнє більш стосується класифікаційних та спеціалізованих прогностичних моделей.

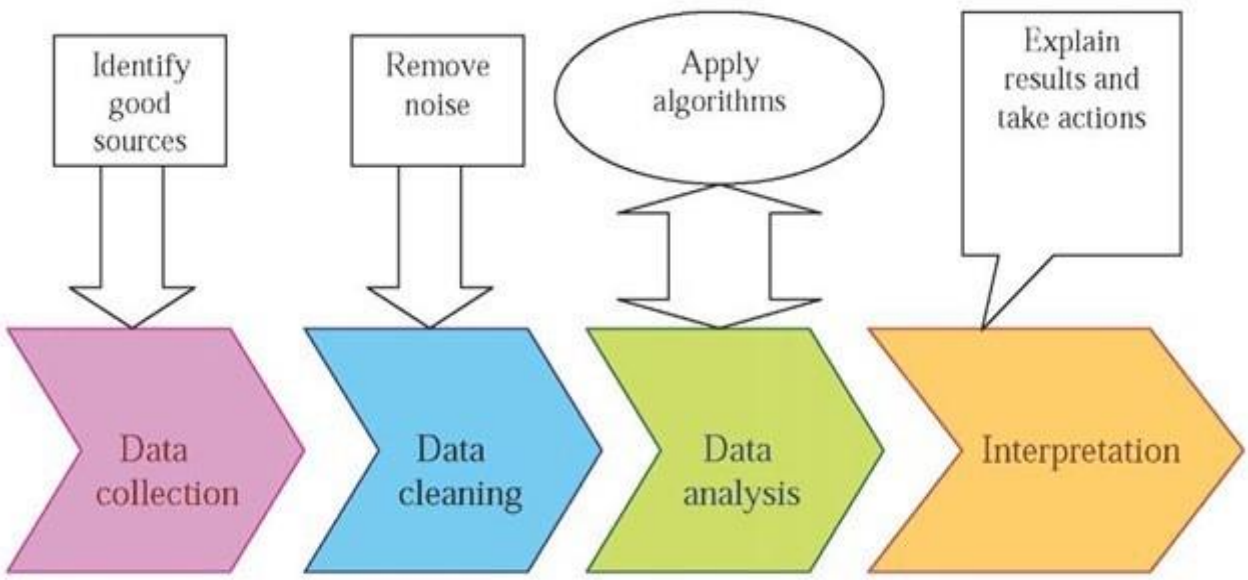


Рисунок 1.1. Класи задач Data Mining.

На рис. 1.2 зображені класи задач Data Mining.

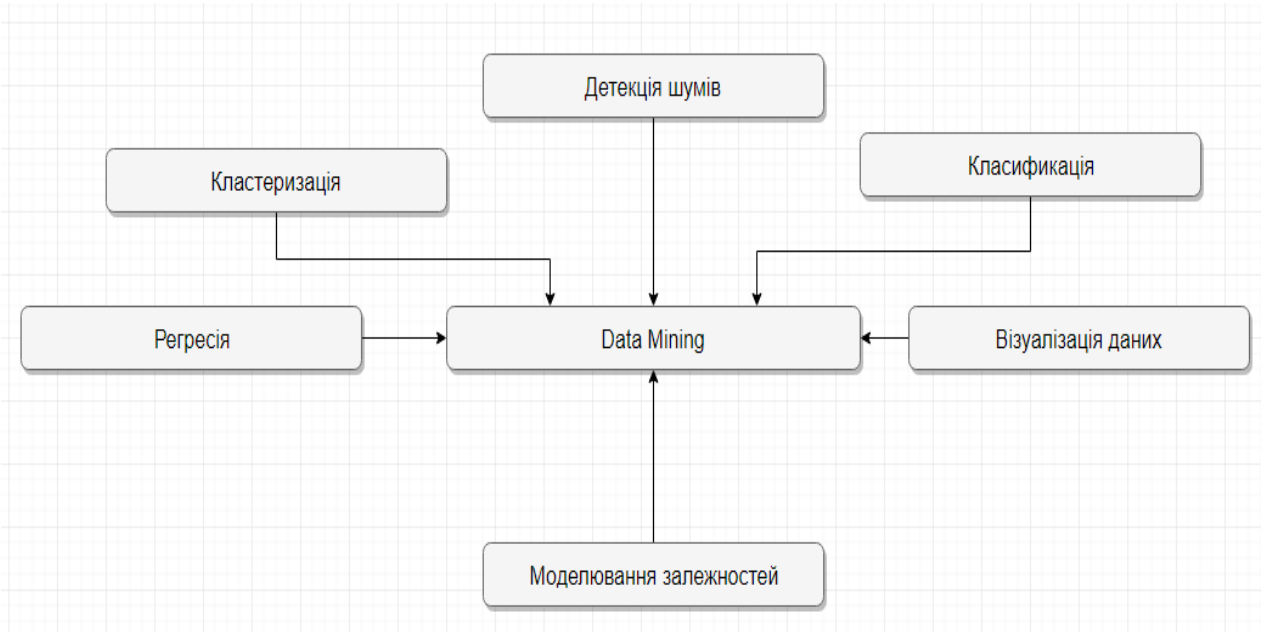


Рисунок 1.2. Класи задач Data Mining.

Розглянемо кожен з цих класів більш детально:

- Детекція (виявлення) шумів - Виявлення незвичайних записів даних (які можуть бути як унікальними випадками, так і помилками), що потребують подальшого дослідження.

- Моделювання залежностей - пошук відношень між змінних. Наприклад, супермаркет може збирати дані про споживацькі звички покупців. За допомогою моделювання залежностей, супермаркет може визначити, які товари часто купуються разом, і використовувати цю інформацію для маркетингових цілей. Це іноді називають аналізом ринкової корзини.
- Кластеризація - це завдання виявлення груп та структур у даних, які в тій чи іншій мірі "подібні", без знання про існуючі структури у даних, що аналізуються.
- Класифікація - це завдання узагальнення відомої структури для застосування до нових даних. Наприклад, фільтр електронної пошти може намагатися класифікувати електронне повідомлення як нормальне або як спам.
- Регресія – задача знаходження функції, яка моделює дані з найменшою помилкою, тобто для оцінки співвідношень між даними або наборами даних.
- Візуалізація даних - надання більш компактного та інформативного представлення набору даних.

1.2. Кластерний аналіз

Кластерний аналіз– це задача розбиття множини об'єктів на групи, які називаються кластерами [1, 2]. Усередині кожної групи повинні виявитися «схожі» об'єкти [3. 4. 5. 6], а об'єкти різних групи повинні бути якомога більш відмінні. Кластеризацію не слід плутати з класифікацією, оскільки вона вирішує задачу розбиття на групи, а не прогнозування. Крім того кластеризація не потребує навчальної вибірки. Такий підхід називається навчанням без вчителя і використовується тоді, коли додаткові відомості про наявні у вибірці групи даних відсутні.

Кластерний аналіз виконує такі основні завдання:

- розробка типології або класифікації;
- дослідження корисних концептуальних схем групування об'єктів;
- породження гіпотез на основі дослідження даних;
- перевірка гіпотез або дослідження для визначення, чи дійсно типи (групи), виділені тим чи іншим способом, присутні в наявних даних.

Застосування кластерного аналізу в загальному вигляді зводиться до наступних етапів:

1. Відбір вибірки об'єктів для кластеризації.

2. Визначення множини змінних, за якими будуть оцінюватися об'єкти у вибірці та нормалізація даних.

3. Обчислення значень обраної міри відстані між об'єктами.

4. Застосування методу кластерного аналізу для створення груп схожих об'єктів (кластерів).

5. Представлення результатів аналізу.

Після отримання та аналізу результатів можливе корегування обраної метрики, методу кластеризації, параметрів алгоритму до отримання оптимального результату.

Типовий результат роботи кластеризаційних алгоритмів зображено на рис. 1.3.

Кластерний аналіз застосовується у багатьох наукових галузях. У біології кластеризація використовується дуже широко. Перш за все це стосується ієрархічних алгоритмів, так званих алгоритмів таксономії. У біоінформатиці алгоритми таксономії використовуються для аналізу складних мереж взаємодіючих генів та побудови філогенетичних дерев. У екології кластеризацію використовують для виділення просторово однорідних груп та спільнот. У соціології також широко використовуються методи ієрархічної кластеризації.

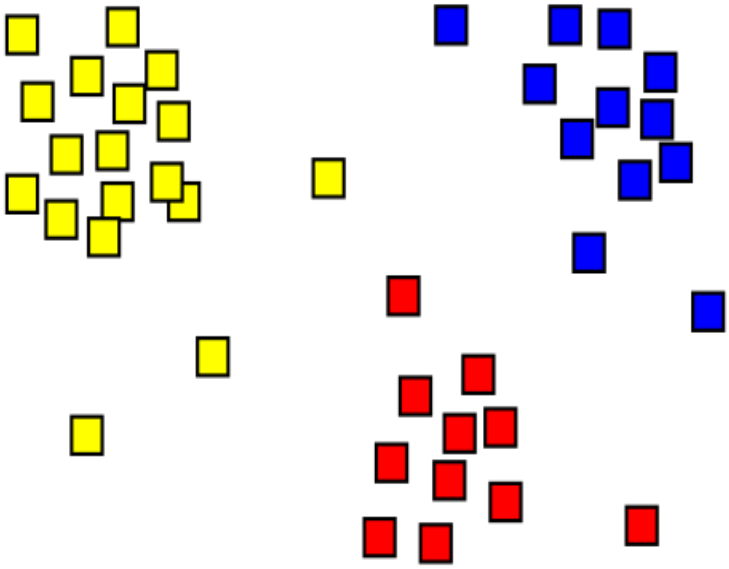


Рисунок 1.3. Приклад результату кластеризації.

У інформатиці кластерний аналіз використовується для кластеризації результатів пошуку, щоб користувач міг отримати максимально релевантний результат. Крім того кластеризація широко застосовується для сегментації зображень. Особливо широко для цього використовують алгоритми, що базуються на нейронних мережах та алгоритми подібні до нечіткого С-середніх. Крім того, кластеризація використовується як один із етапів інтелектуального аналізу даних (Data Mining) для роботи із групами схожих об'єктів замість аналізу всієї вібірки.

1.3. Постановка задачі кластеризації

Нехай об'єкт $\boldsymbol{o}$ вихідної матриці X типу "об'єкт-властивості" розміром $n \times m$ описується вектором $\langle X_1, X_2, \dots, X_m \rangle$ та може бути представлений у вигляді точки $\boldsymbol{o_i} = \langle x_{i1}, x_{i2}, \dots, x_{im} \rangle$. Також нехай задана міра відстані між

двома об'єктами $d_{ij} = d(\boldsymbol{o}_i, \boldsymbol{o}_j)$ та множина класів $C = \{c_1, c_2, \dots, c_k\}$. Задачею кластеризації тоді буде віднести кожен об'єкт $\boldsymbol{o}_i$ з класом з C.

Постановка задачі може змінюватись для різних підходів до кластеризації. Наприклад, при нечіткій кластеризації один і той самий об'єкт може з різним ступенем належати до різних кластерів.

1.4. Алгоритми кластеризації

Виділяють дві основні класифікації алгоритмів кластеризації.

І. Ієрархічні та не ієрархічні.

Ієрархічні алгоритми (також відомі як алгоритми таксономії) будують не одне розбиття вибірки на непересічні кластери, а систему вкладених розбиттів [7, 8]. У ході роботи таких алгоритмів будується дерево розбиття (дендрограма) (рис. 1.4), коренем якого є вся вибірка, а листям – кластери, що складаються з одного об'єкта [8, 9].

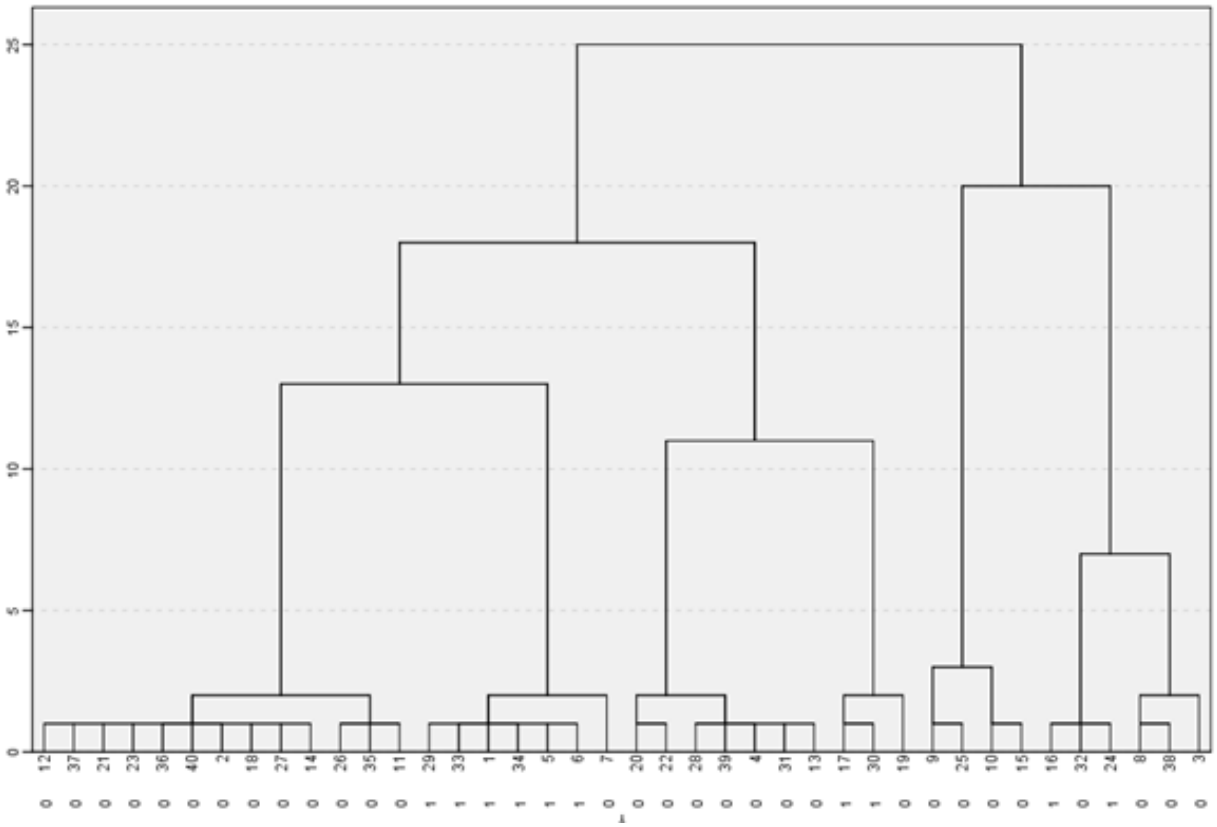


Рисунок 1.4. Приклад дерева розбиття.

Серед алгоритмів ієрархічної кластеризації виділяються два основних типи: агломеративні і дивізимні алгоритми [10].

Дивізимні алгоритми працюють за принципом «зверху-вниз»: на початку роботи алгоритму всі об'єкти належать до одного кластера і в ході роботи розбиваються на більш дрібні кластери, доки кожен об'єкт не буде належати окремому кластеру.

Більш поширені агломеративні алгоритми, які на початку роботи поміщають кожен об'єкт в окремий кластер, а потім об'єднують кластери в усе більші, поки всі об'єкти вибірки не будуть об'єднані в один великий кластер.

У разі використання ієрархічних алгоритмів постає питання, як об'єднувати між собою кластера, як обчислювати «відстані» між ними.

Існує кілька метрик:

1. Метод найближчого сусіда.

У даному методі відстань між двома кластерами визначається як відстань між двома найбільш близькими, за вибраною мірою, об'єктами (найближчими сусідами) в різних кластерах. Результуючі кластери при такому підході мають тенденцію об'єднуватися в ланцюжки (рис. 1.5).

2. Повний зв'язок (відстань найбільш віддалених сусідів).

У цьому методі відстані між кластерами визначаються вже найбільшою відстанню між будь-якими двома об'єктами в різних кластерах (тобто найбільш віддаленими сусідами).

Цей метод зазвичай добре показує себе, коли об'єкти походять з окремих груп. Якщо ж кластери мають подовжену (стрічкоподібну) форму або їх природний тип є «ланцюговий», то цей метод є непридатним.

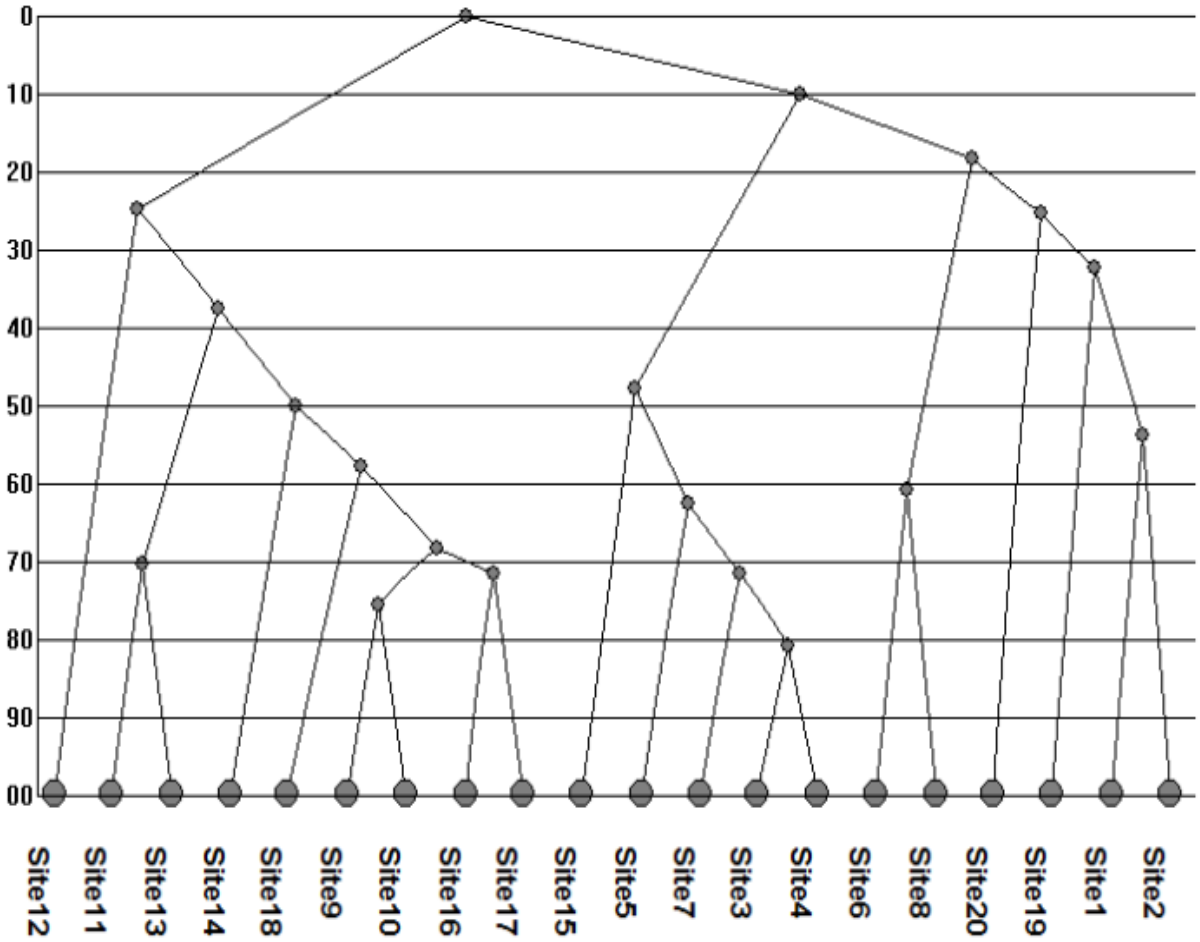


Рисунок 1.5. Ланцюжки при методі найближчого сусіда.

Приклад побудови розбиття методом повного зв'язку зображено на рис. 1.6.

3. Незважене попарне середнє.

У цьому методі відстань між двома різними кластерами обчислюється як середня відстань між усіма парами об'єктів в них. Метод ефективний, коли об'єкти формують різні групи, однак він працює однаково добре і в випадках стрічкоподібних кластерів. Приклад роботи методу зображено на рис. 1.7 Метод широко використовується у біоінформатиці для побудови філогенетичних дерев.

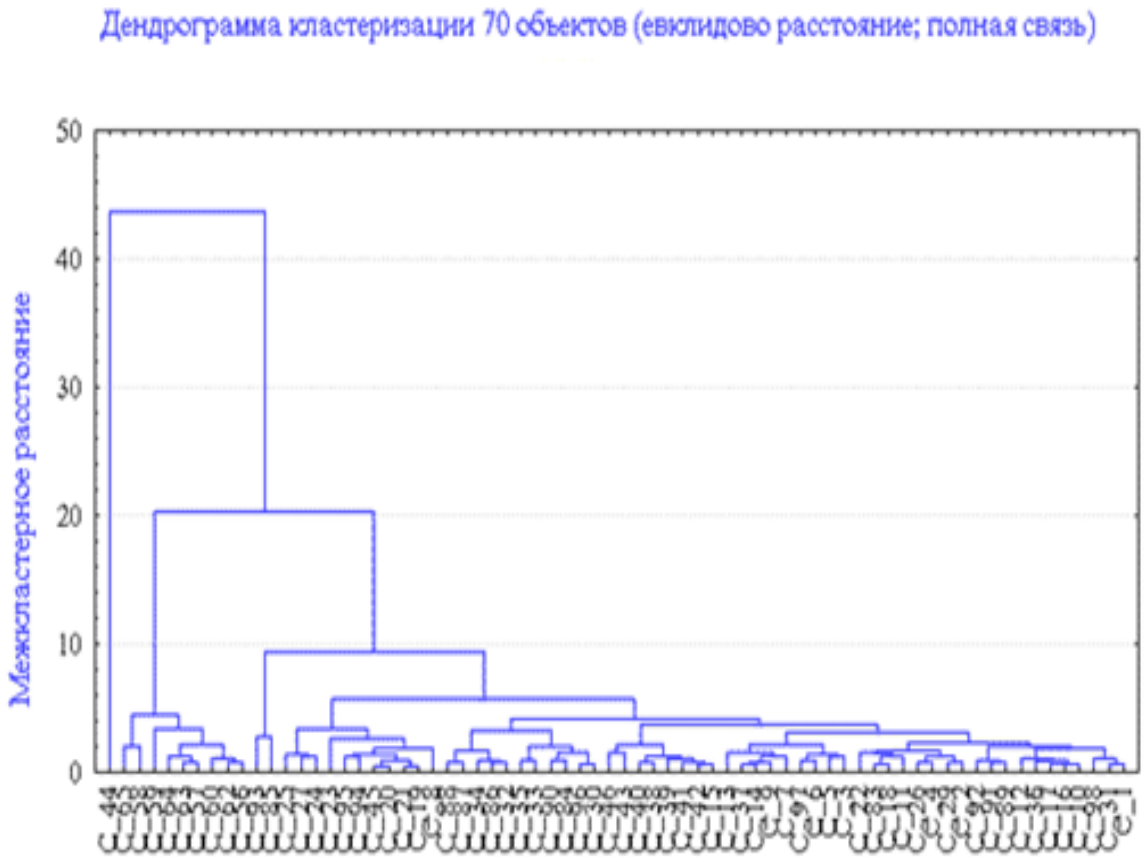


Рисунок 1.6. Метод повного зв'язку.

4. Зважене попарне середнє.

Метод ідентичний методу незваженого попарного середнього, за винятком того, що при обчисленнях розмір кластерів (тобто число об'єктів, що містяться в них) використовується в якості вагового коефіцієнта. Тому доречно використовувати даний метод у тих випадках, коли відомо, що кластери сильно відрізняються за розміром.

5. Незважених центроїдний метод.

У цьому методі відстань між двома кластерами визначається як відстань між їх центрами мас.

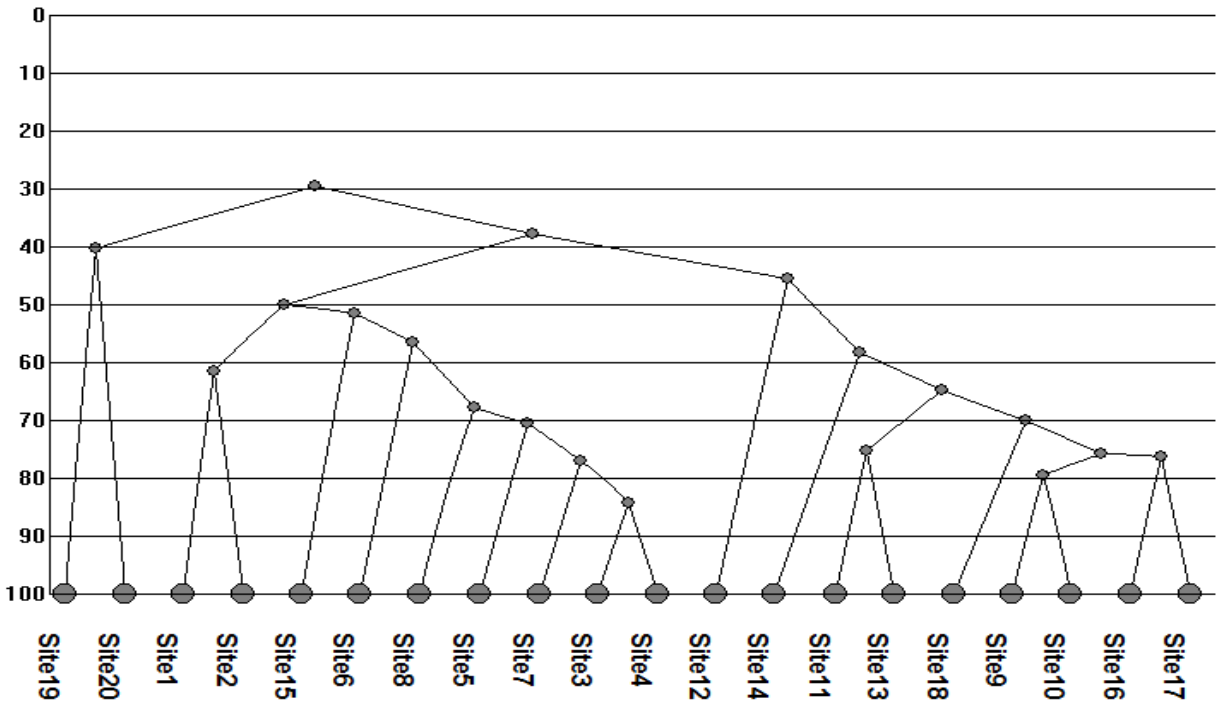


Рисунок 1.7. Метод незваженого попарного середнього.

6. Зважений центроїдний метод (медіана).

Цей метод ідентичний попередньому, за винятком того, що при обчисленнях використовуються вагові коефіцієнти для урахування різниці між розмірами кластерів. Тому цей метод підходить у тих випадках, коли підозрюють, що кластери будуть сильно відрізнятись за розміром.

7. Метод Варда (Уорда).

Метод мінімальної дисперсії Уорда - це особливий випадок об'єктивного функціонального підходу, який спочатку був представлений Джо Х. Уордом-молодшим [11]. Уорд запропонував загальну агломеративну процедуру ієрархічної кластеризації, де критерій вибору пари кластерів для об'єднання на кожному кроці ґрунтується на оптимальному значенні об'єктивної функції. Ця цільова функція може мати будь-який вигляд, залежно від вирішуваної задачі. Для ілюстрації процедури Уорд використовував приклад, де цільова функція є сумою квадратів залишків, і цей приклад називається методом Уорда або, точніше, методом мінімальної дисперсії Уорда.

Критерій мінімальної дисперсії Уорда мінімізує загальну дисперсію всередині кластера. Щоб реалізувати цей метод, на кожному кроці необхідно знайти пару кластерів, що призведе до мінімального збільшення загальної дисперсії всередині кластера після об'єднання. Це збільшення є зваженим квадратом відстані між кластерними центрами. На початковому етапі всі кластери містять лише одну точку. Щоб застосувати рекурсивний алгоритм до цієї цільової функції, початкова відстань між окремими об'єктами повинна бути (пропорційна) квадрату евклідової відстані. Приклад роботи алгоритму забражено на рис. 1.8.

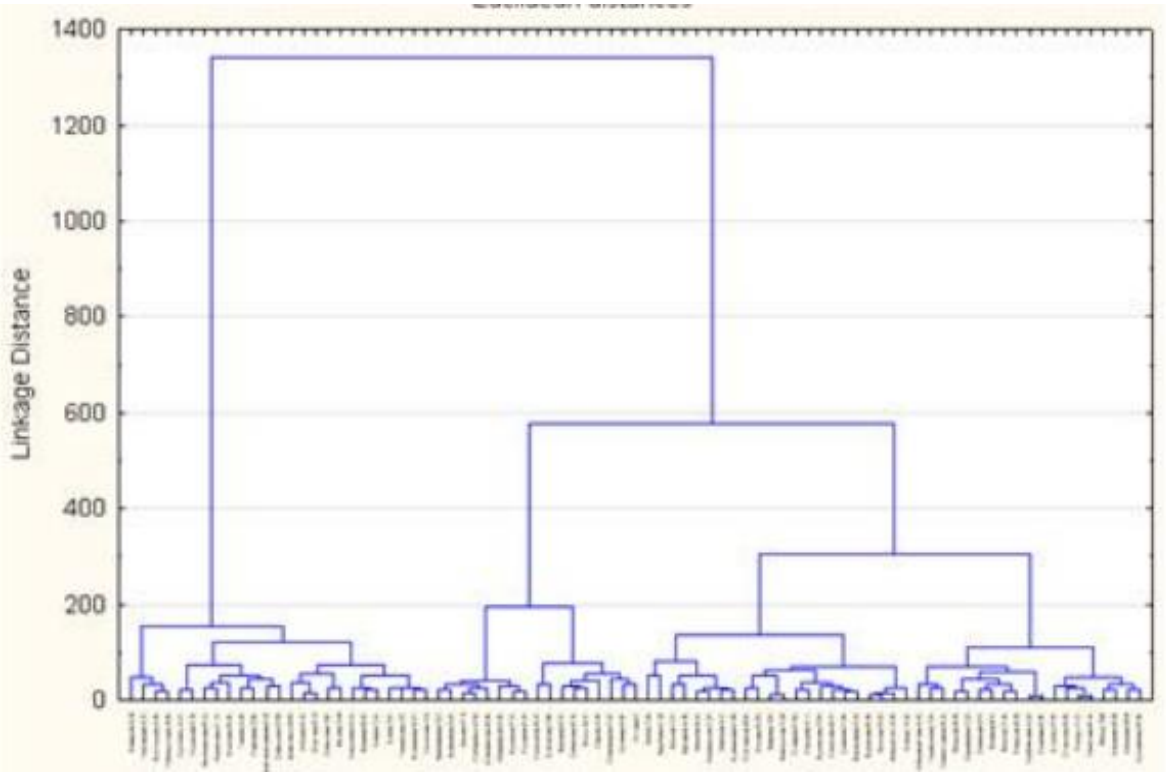


Рисунок 1.8. Метод Уорда.

Як показано на рис. 1.9, результати кластеризації методами таксономії незважаючи на високу концептуальну подібність можуть давати результат, що мало співпадає. Тому при використанні ієрархічних алгоритмів слід здійснювати вибір методі залежно від очікуваного змісту отриманих у результаті кластеризації груп.

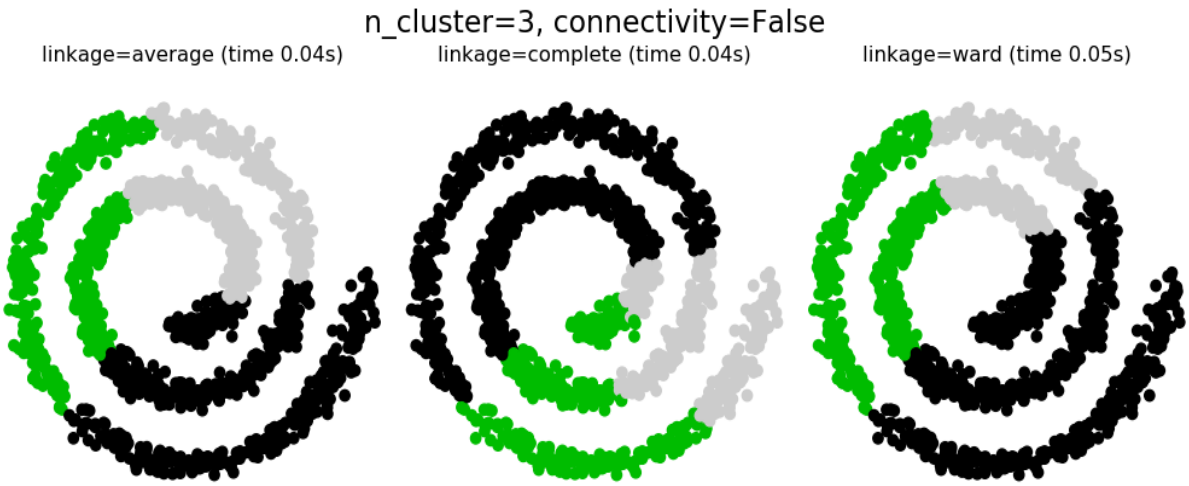


Рисунок 1.9. Порівняння результатів роботи методів ієрархічної кластеризації.

Серед не ієрархічних алгоритмів кластеризації найбільшу увагу викликають ітеративні алгоритми та алгоритми, що основуються на щільності.

Ітеративні алгоритми кластеризації називають так тому, що вони циклічно перерозподіляють об'єкти між кластерами. Одним із найвідоміших представників таких алгоритмів є k- середніх [8, 12, 13]. Головною ідеєю цього алгоритму є мінімізація відстаней між об'єктами в кластерах. В алгоритмі k- середніх мінімізується функція:

$$J = \sum_{k=1}^{M} \sum_{i=1}^{N} d^2(x_i, c_k),$$

де $x_i \in X$ - об'єкт кластеризації, $c_k \in C$ -центр мас кластера (центроїд), а d - міра відстані. Для роботи алгоритму необхідно знати кількість кластерів. На початку роботи алгоритму задається положення центроїдів кластерів. Всі точки приєднуються до того кластера, центроїд якого є найближчим за обраною мірою відстані. Після того, як всі точки приєднані

до кластерів, розраховується нове положення центроїдів і все починається спочатку. Алгоритм завершується коли центроїд не змінить свого положення на початку нової ітерації. Приклад роботи алгоритму зображено на рис. 1.10.

Хоча k-середніх є досить простим та зрозумілим, але він має ряд недоліків. По-перше, алгоритм формує кластери сферичної форми, а по друге, алгоритм є чутливим до викидів.

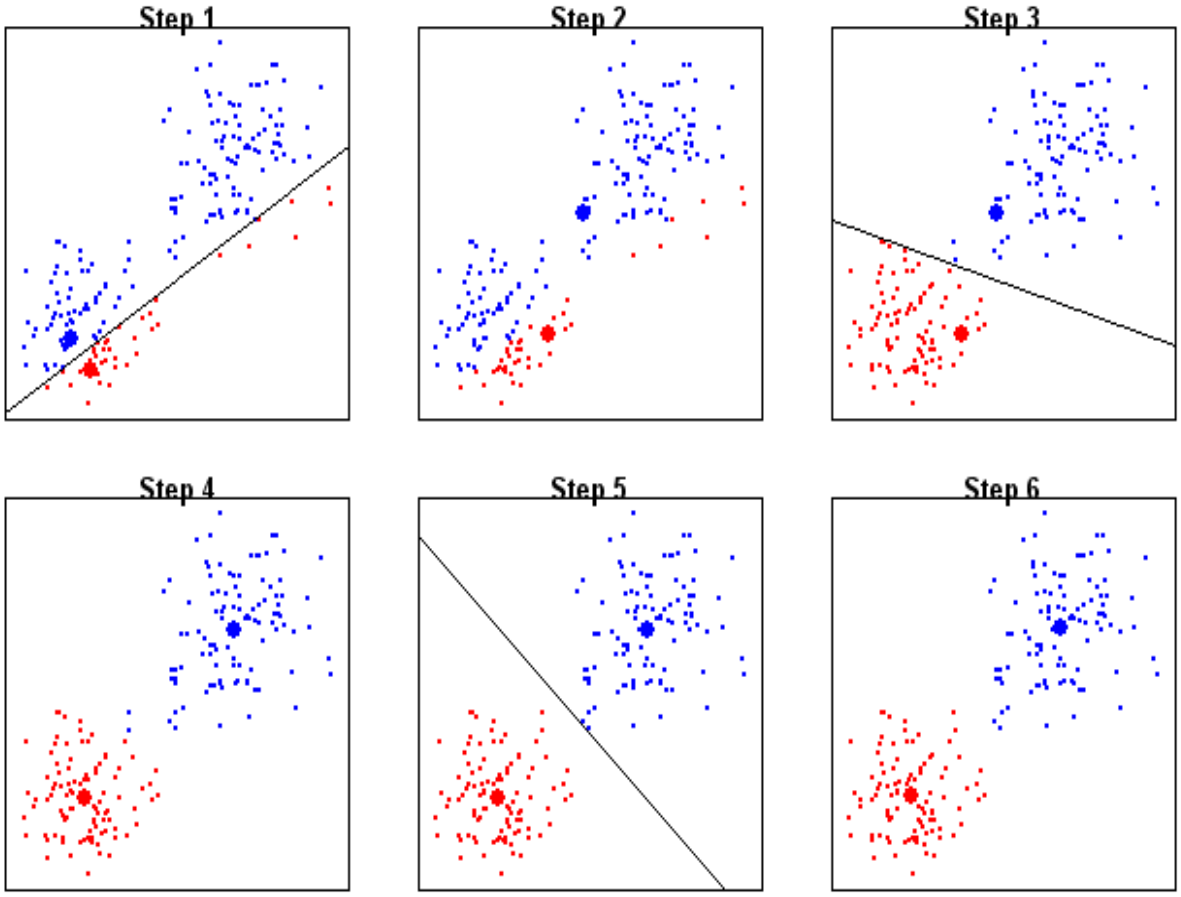


Рисунок 1.10. Приклад роботи алгоритму k-середніх.

Не слід також забувати про те, що для алгоритму k-середніх велике значення має вибір початкового положення центроїдів. Різні методи вибору центроїдів можуть призвести до різних результатів побудови розбиття і у деяких випадках вибір процедури ініціалізації центроїдів може призвести до незадовільного результату (рис. 1.11).

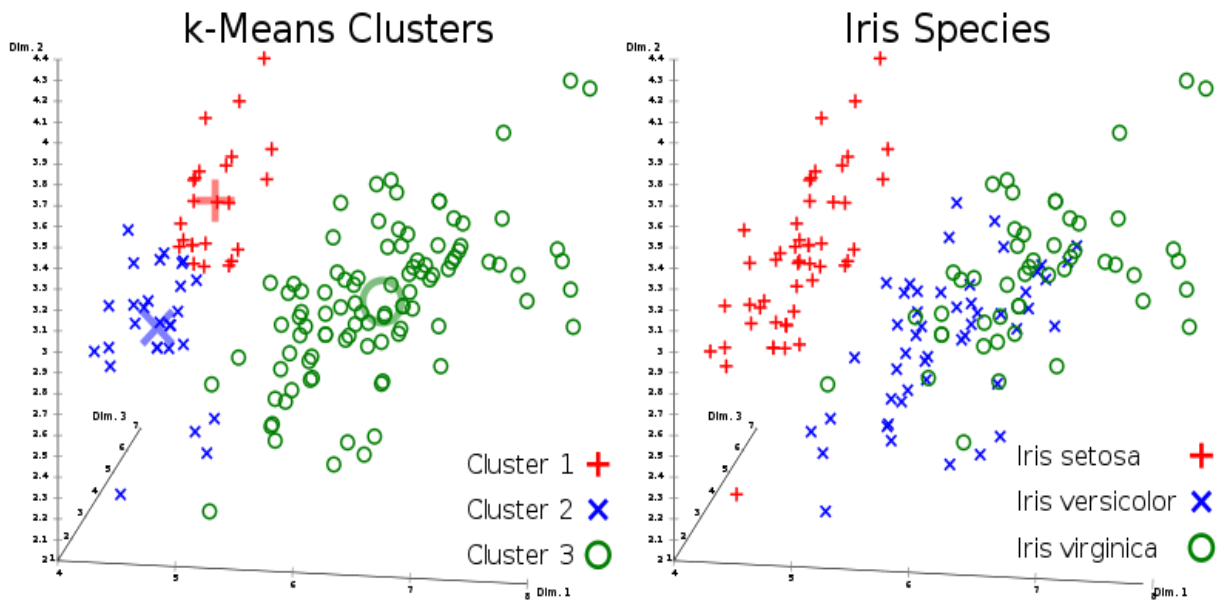


Рисунок 1.11. Приклад невдалих початкових центроїдів у k-середніх.

Алгоритми, що базуються на щільності, визначають кластер як групу об'єктів, що щільно розташовані. Під щільністю в даному випадку мається на увазі те, що в деякому околі точки є деяка мінімальна кількість об'єктів. Класичним представником таких алгоритмів є алгоритм DBSCAN [15, 16]. Абревіатура DBSCAN розшифровується як Density Based Scanning, тобто сканування, що засноване на щільності.

Алгоритм працює наступним чином:

1) Вибирається окіл ε та кількість об'єктів Mpts, що має знаходитись у цьому околі.

2) Береться довільний ще не оброблений об'єкт. Для нього перевіряється умова, що в ε- околі точки є деякий мінімум інших об'єктів: $d(x_i, x_j)<\varepsilon$ для деякої кількості $j>M_{pts}$. Якщо це не так, то ця точка є шумом і ті самі дії повторюються для наступної точки.

3) Якщо умова виконується, то точка помічається як та, що належить кластеру. Це так звана коренева точка. Точки. що знаходяться навколо неї, заносяться в окрему категорію.

4) Кожна не оброблена точка з цієї категорії спочатку помічається як та, що належить кластеру, а потім перевіряється, що в епсилон-околі точки є деякий мінімум інших об'єктів d(xi, xj)<ε для деякої кількості j>Mpts. Якщо це так, то ці точки заносяться в цю ж категорію.

5) Після перевірки точка виноситься з цієї тимчасової категорії. Очевидно, що рано чи пізно точки в даній категорії закінчаться (на межі кластера умова наявності об'єктів в околі не буде виконуватись). Тоді переходимо до кроку 2. Інакше повертаємося до кроку 4.

Приклад результату роботи алгоритму для кластерів різної форми зображено на рис. 1.12. Як можна бачити, алгоритм добре розпізнає як сферичні кластери, так і кластери стрічкоподібної форми.

Очевидним недоліком цього алгоритму є те, що він не може зв'язати кластери через вузькі місця, де правило щільності не буде виконуватись. Крім того недоліком також є те, що для застосування алгоритма необхідно додатково провести анализ вхідних даних. Алгоритм має більше ступенів свободи, ніж k-середніх і якщо у випадку k-середних можна задавати велику кількість кластерів і зменшувати її шляхом проведення повторної кластеризації, то для вибору оптимальних параметрів у випадку DBSCAN необхідно вводити алгоритм аналізу вхідних даних з урахуванням специфіки задачі, що вирішується.

Впорядковування точок для ідентифікації кластерної структури (OPTICS) - це алгоритм для пошуку кластерів [16], заснований на щільності, в просторових даних. Його розробили Міхаель Анкерст, Маркус М. Брейніг, Ганс-Петро Крігель та Йорг Сандер. [17] Його основна ідея схожа на DBSCAN , однак вона спрямована на один з головних недоліків DBSCAN: проблему виявлення значущих кластерів в даних різної щільності. Для цього точки у базі даних упорядковані таким чином, що точки, які є просторово найближчими, стають сусідами у загальному порядку. Крім того, спеціальна відстань зберігається для кожної точки, яка представляє собою щільність,

яка має бути прийнята для кластера, щоб обидві точки належати до одного кластеру.

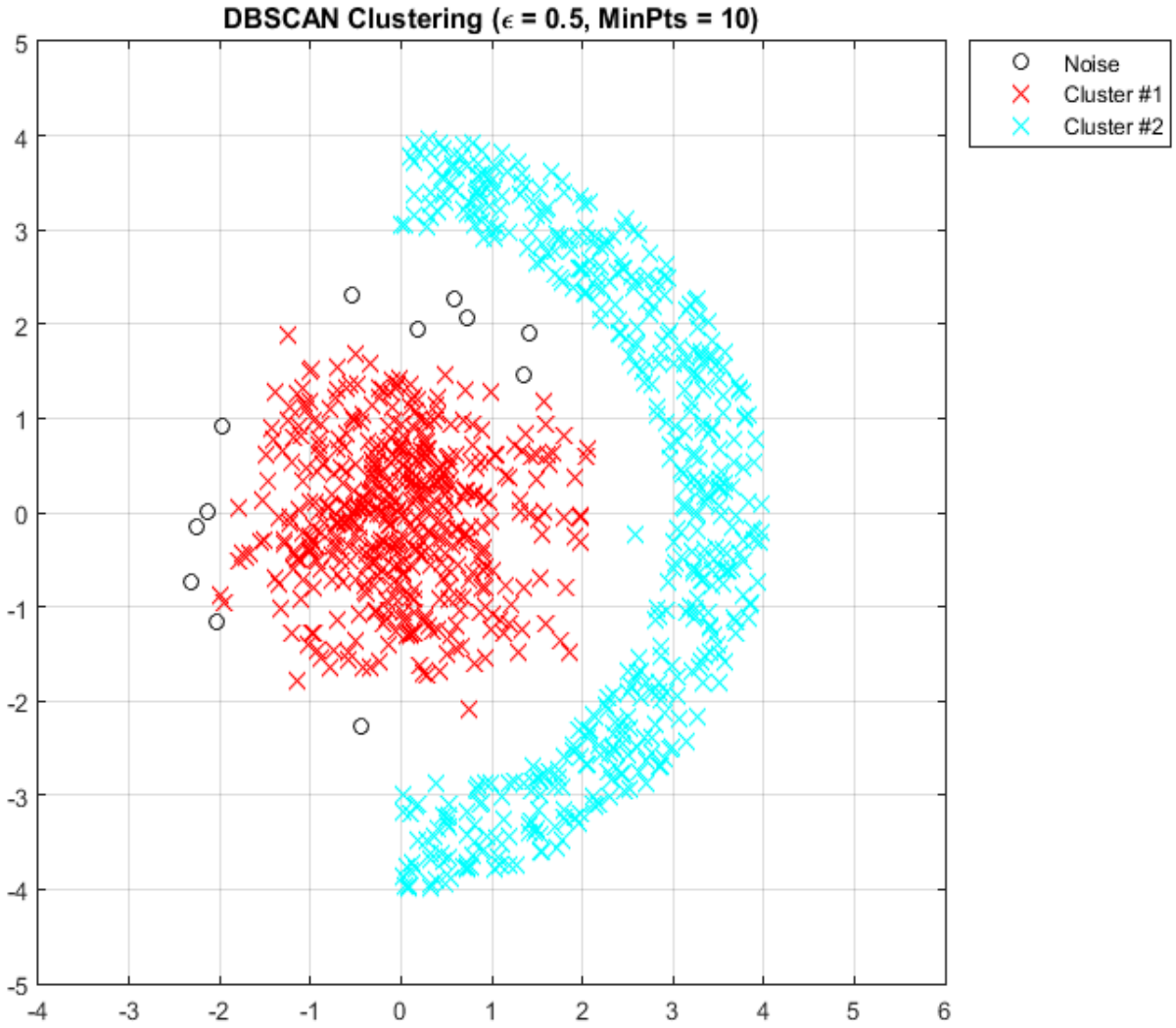


Рисунок 1.12. Приклад результату роботи алгоритму DBSCAN.

. Як і DBSCAN, OPTICS вимагає двох параметрів: ε, який описує максимальну відстань (радіус) для розгляду, і MinPts, що описує кількість точок, необхідних для створення кластера. Точка p є основною точкою, якщо щонайменше MinPts точок знаходяться в його ε-околі. На відміну від DBSCAN, OPTICS також розглядає точки, які є частиною більш щільно упакованого кластера, тому кожній точці присвоюється ядерна відстань (core distance), яка описує відстань до найближчої точки MinPtsth.

Відстань досяжності іншої точки o від точки p - це або відстань між o і p, або ядерна відстань p, залежно від того, що більше.

Якщо p та o є найближчими сусідами, то необхідно зробити висновок що ε ε`< ε, щоб p та o належали до одного кластера.

Як ядерна відстань, так і відстань досяжності невизначені, якщо немає достатньо щільного кластеру. З огляду на досить великий ε, це ніколи не відбудеться, але тоді кожен запит з ε-околу поверне усю базу даних. Отже, параметр ε необхідний для відсічення щільності кластерів, що більше не вважаються релевантними, і таким чином прискорює алгоритм.

Використовуючи графік досяжності (спеціальний вид дендрограми), легко можна отримати ієрархічну структуру кластерів. Це 2D графік, з упорядкуванням точок, оброблених OPTICS по осі x і відстані досяжності на осі Y. Оскільки точки, що належать кластерові, мають низьку відстань досяжність до свого найближчого сусіда, кластери відображаються як долини на графіку доступності. Чим глибша долина, тим щільніше кластер.

Рисунок 1.13 ілюструє цю концепцію. У верхній лівій області показаний набір даних із штучного прикладу. У верхній правій частині візуалізується дерево покриття, створене OPTICS, а нижня частина показує графік досяжності, який обчислюється за допомогою OPTICS. Кольори на цьому графіку є мітками, а не обчислені алгоритмом; але добре видно, як долини на графіку відповідають кластерам у наведеному вище наборі даних. Жовті точки в цьому зображенні вважаються шумом, і не мають долини на графіку досяжності. Вони, як правило, не призначаються для кластерів, за винятком завжди присутнього кластера, що об’єднує усі дані, в ієрархічному результаті.

Вилучення кластерів з графіка можна зробити вручну, вибравши діапазон на осі x після візуального огляду, вибравши порогові значення на осі y (результат буде аналогічним результату кластеризації DBSCAN з тими ж параметрами) або різними алгоритмами, які намагаються виявляти долини за крутизною, виявленням коліна або локальними максимумами.

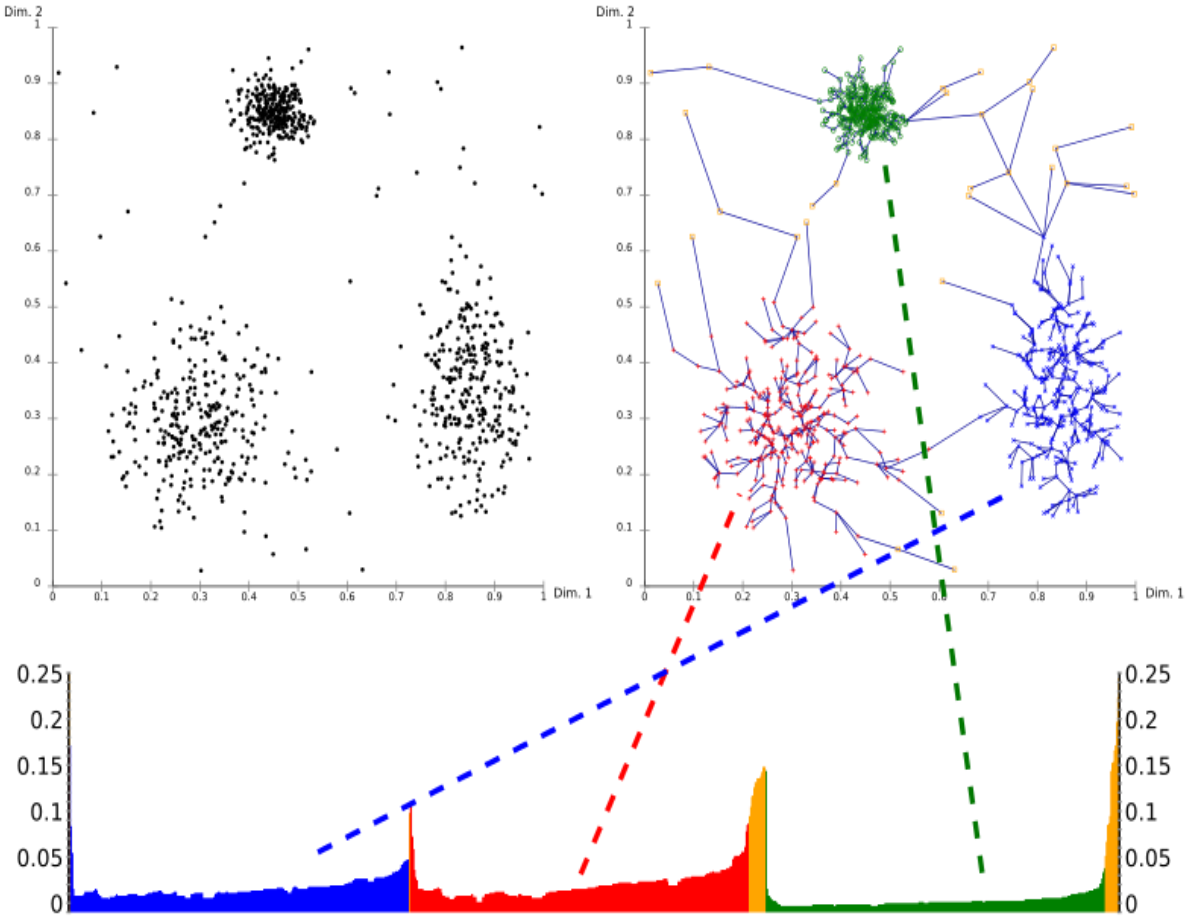


Рисунок 1.13. Приклад роботи алгоритму OPTICS.

Кластери, отримані таким чином, зазвичай ієрархічні, і не можуть бути отримані за допомогою одного циклу DBSCAN.

Нейронні мережі Кохонена типовий приклад архітектури нейронної мережі, яка навчається без учителя [18]. Звідси випливає перелік вирішуваних мережею завдань: кластеризація даних або прогнозування їх властивостей. Крім того, мережі Кохонена можуть використовуватися з метою зменшення розмірності даних з мінімальною втратою інформації.

Здебільшого архітектури нейронних мереж навчаються з учителем на еталонних вибірках даних, що включають набір прикладів, що складаються з відповідних один одному пар вхідних і вихідних векторів. При цьому вихідні значення безпосередньо беруть участь в обчисленні вагових коефіцієнтів. У нейронних мережах Кохонена вихідні вектори в навчальних

даних можуть як бути, так і бути відсутніми, і, в будь-якому випадку, вони не беруть участі в процесі навчання. Тобто виходи не використовуються в якості орієнтирів при корекції синапсів. Саме тому даний принцип настройки нейронної мережі називається самонавчанням.

У розглянутій архітектурі сигнал поширюється від входів до виходів в прямому напрямку. Структура нейронної мережі містить єдиний шар нейронів (шар Кохонена) без коефіцієнтів зміщення (рис. 1.14). Загальна кількість вагових коефіцієнтів розраховується як добуток:

$$N_w = MK,$$

де M – розмір вхідного вектора, K – кількість нейронів.

Кількість нейронів дорівнює кількості кластерів, серед яких відбувається початковий розподіл і подальший перерозподіл навчальних прикладів. Кількість вхідних змінних нейронної мережі дорівнює числу ознак, що характеризують об'єкт дослідження і на основі яких відбувається віднесення його до одного з кластерів.

Слід розрізняти власне самонавчання і самоорганізацію нейронної мережі Кохонена. При звичайному самонавчанні мережа має строго фіксовану структуру, тобто кількість нейронів, що не змінюється протягом усього життєвого циклу. При самоорганізації мережа, навпаки, не має постійної структури. Залежно від знайденої відстані до нейрона-переможця або цей нейрон використовується для кластеризації прикладу, або для поданого на входи прикладу створюється новий кластер з відповідними йому ваговими коефіцієнтами. Крім того, в процесі самоорганізації структури мережі Кохонена окремі нейрони можуть виключатися з неї.

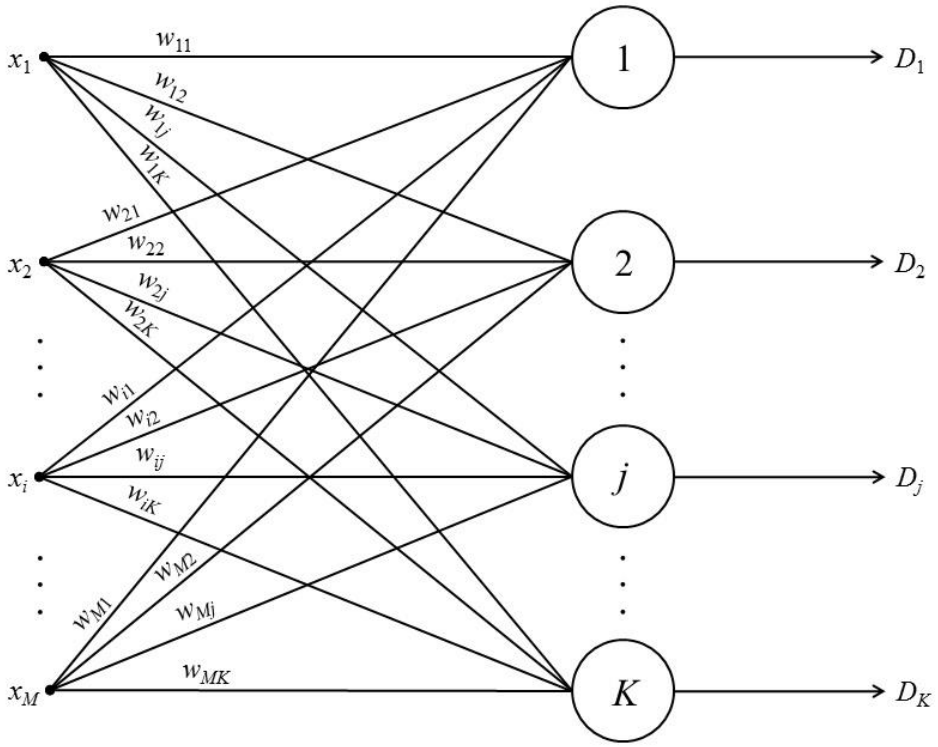


Рисунок 1.14. Структура нейронної мережі Конохена.

Для навчання мережі Кохонена використовується змагальний метод [18, 19]. На кожному кроці навчання з вихідного набору даних випадково вибирається один вектор. Потім проводиться пошук нейрона вихідного шару, для якого відстань між його вектором ваг і вхідним вектором є мінімальною.

Алгоритм навчання мережі Конохена працює наступним чином [20]:

1) Ініціювати матрицю ваг малими випадковими значеннями (на відрізку [-1,1]).

2) Побудувати чергу з елементів вхідної множини, розташувавши їх у випадковому порядку, позначити їх всі як необроблені.

3) Вибрати перший необроблений елемент *x* з черги.

4) Для кожного виходу j обчислити відстань *dj* між його вектором ваг *wj* і вхідним вектором *x*:

$$d_j = p(w_j, x) .$$

5) Знайти номер вихідного нейрона *jm* з мінімальним відстанню *dj:*

$$j_m = \arg\min_j (d_j).$$

6) Обчислити зміну ваг ΔW = {Δw_u} для всіх нейронів *u* вихідного шару:

$$\Delta \overline{w}_u = (\overline{w}_u - \overline{x}) \cdot h(u, c, t) \cdot \eta ,$$

де *c* - номер (пара індексів) нейрона переможця j_m в двовимірної решітці другого шару;

u - номер (пара індексів) нейрона з вектором ваг wu в двовимірної решітці другого шару;

$\overline{w}_u$ - вектор вагових коефіцієнтів зв'язку вхідного шару і вихідного нейрона номер u;

$\bar{x}$ - поточний вектор входів мережі;

h (u, c, t) - значення функції околу для нейрона номер u в момент часу t;

η - коефіцієнт швидкості навчання;

7) Скорегувати матрицю ваг:

$$W = W - \Delta W .$$

8) Позначити елемент вхідної черги x як оброблений.

9) Якщо в черзі залишаються не оброблені точки то перейти на п. 3.

10) якщо критерій зупинки навчання не досягнуо, то перехід на п. 2.

В якості критеріїв зупинки процесу навчання можна використовувати наступні:

- кількість повних циклів навчання обмежена константою, наприклад кількість циклів дорівнює кількості елементів у вхідній множині;
- вихід мережі стабілізується, тобто вхідні вектори не переходять між кластерними елементами;
- зміни ваг стають незначними.

У разі самоорганізації мережі Кохонена алгоритм зазнає певних змін [21]:

1) Здається критична відстань $R_{кр}$, що відповідає максимально допустимомій евклидовій відстані між входами прикладу і вагами нейрона-переможця. Початкова структура не містить нейронів. При подачі на входи мережі самого першого прикладу навчальної вибірки створюється перший нейрон з ваговими коефіцієнтами, рівними поданням вхідним значенням.

2) На входи мережі подається новий випадково обраний приклад поточної епохи навчання, розраховуються евклідові відстані від прикладу до центру кожного кластера і визначається нейрон-переможець з найменшим з них R_{min}.

3) Якщо виконується умова $R_{min} \leq R_{кр}$, проводиться корекція вагових коефіцієнтів відповідного нейрона-переможця, в іншому випадку в структуру мережі додається новий нейрон, вагові коефіцієнти якого приймаються чисельно рівними вхідним значенням поданого прикладу.

4) Процедура повторюється з п. 2. Якщо протягом останньої епохи навчання будь-які кластери залишилися не задіяними, то відповідні нейрони виключаються зі структури мережі Кохонена.

5) Обчислення закінчуються, якщо виконується одна з умов, прописаних в алгоритмі самонавчання мережі фіксованого структури.

Результат кластеризації за допомогою мережі Конохена зображено на рис. 1.15.

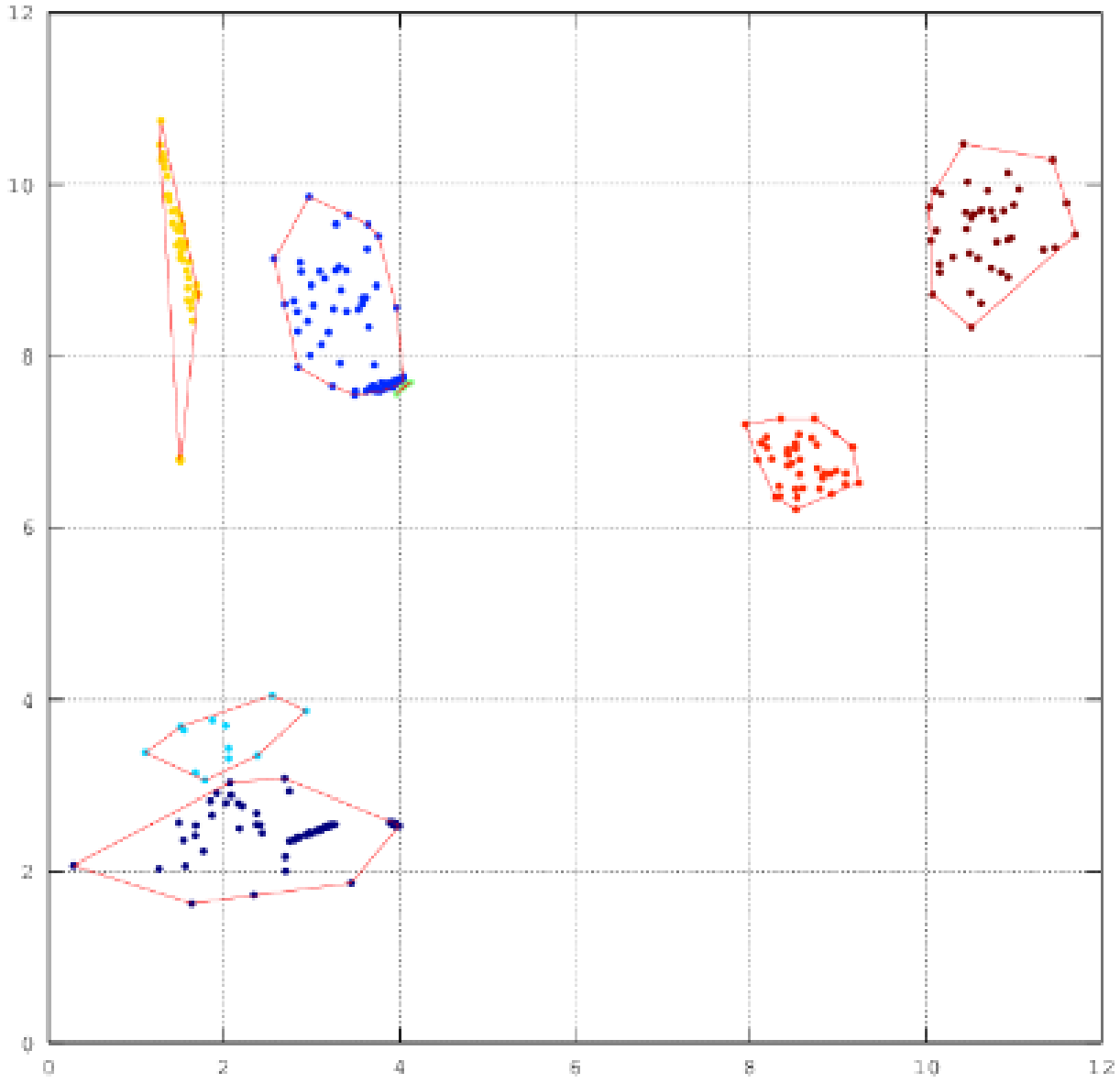


Рисунок 1.15. Приклад результату роботи мережі Конохена.

Алгоритм FOREL був запропонований у 1967 році при вирішенні палеонтологічної задачі. Алгоритм має багато варіацій, що описані у [22]. В основі алгоритму лежить одна базова процедура [23].

Нехай задана деяка точка x0 ∈ X і параметр R. Виділяються всі точки вибірки $x_i \in X_\ell$, що потрапляють всередину сфери $\rho(x_i, X_0) \leq R$, і точка x_0 переноситься в центр мас виділених точок. Ця процедура повторюється до тих пір, поки набір виділених точок, а значить і положення центру, не перестане змінюватися (рис 1.16).

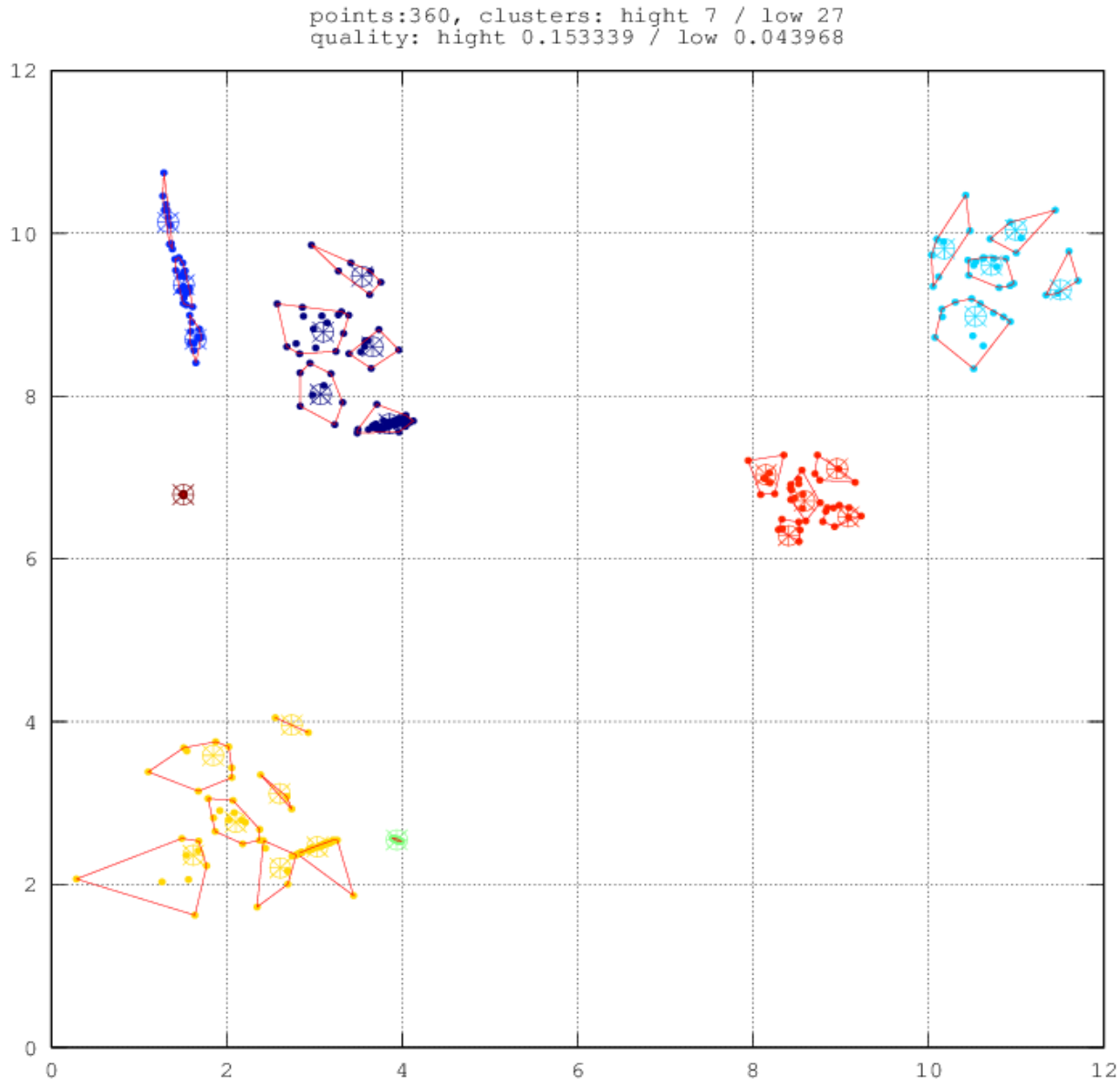


Рисунок 1.16. Результат роботи алгоритму FOREL.

Доведено, що ця процедура сходиться за кінцеве число кроків. При цьому сфера переміщується в місце локального згущення точок. Центр сфери x_0 в загальному випадку не є об'єктом вибірки, тому і називається формальним елементом.

Для обчислення центру необхідно, щоб множина об'єктів X була не тільки метричним, а й лінійним векторним простором. Ця вимога виконується, коли об’єкти описуються числовими ознаками. Однак можливі варіанти завдань, в яких спочатку задана тільки метрика, а додавання і

множення на число не визначені на X. В такому випадку як центр сфери можна вибрати такий об'єкт вибірки, для якого середня відстань до інших об'єктів кластера є мінімальною.

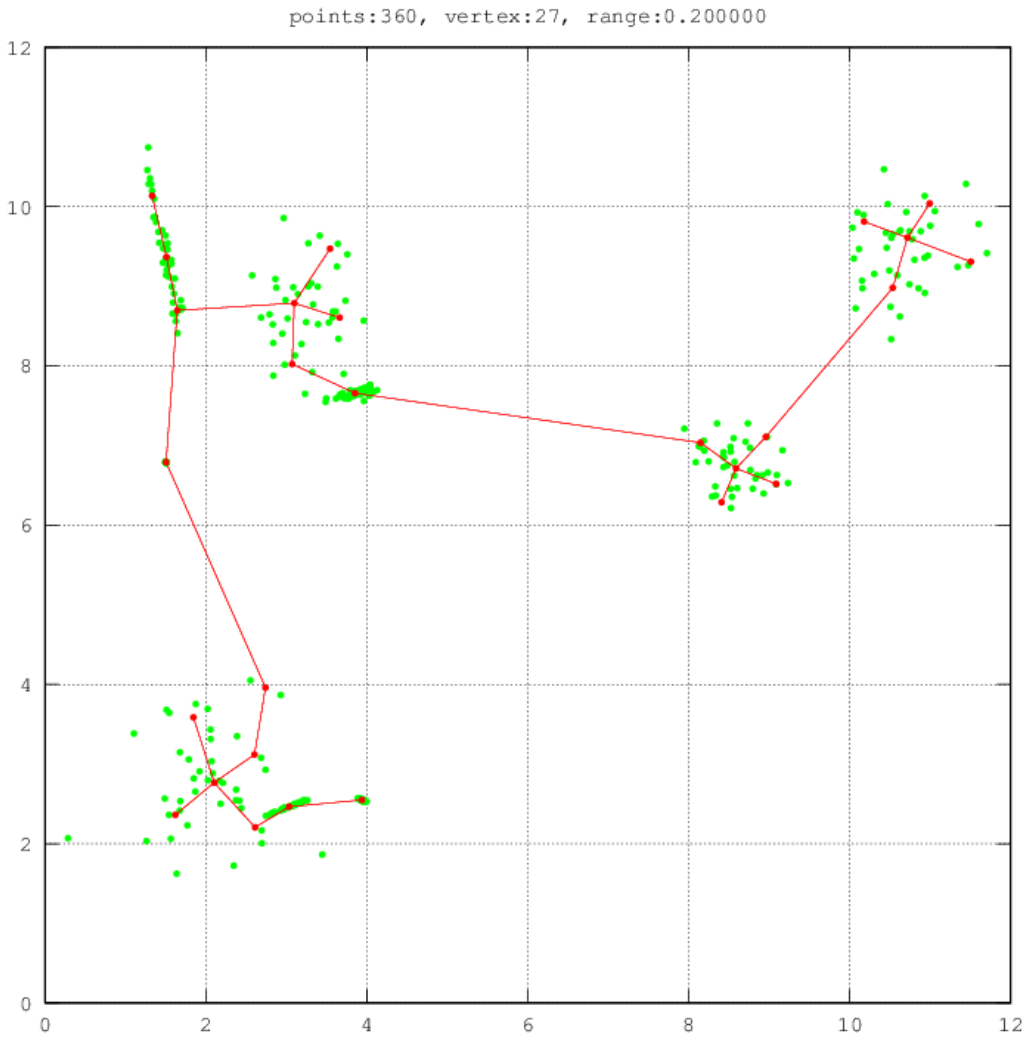


Рисунок 1.17. Застосування найкоротшого незамкненого шляху до результату FOREL.

Даний алгоритм дозволяє описувати кластери довільної форми за рахунок зміни значення параметру R. Для цього необхідно задати досить малий R. Якщо кластери у вибірці близькі до сферичної форми, то можна взяти велике значення R.

Слід також враховувати, що алгоритм є дуже чуттєвим до вибору початкової точки,тому необхідно проводити декілька повторних кластеризацій.

Для об'єднання отриманих у результаті роботи алгоритму сфер у кластери використовують також алгоритми на кшталт найкоротшого незамкненого шляху (рис 1.17). Алгоритм найкоротшого незамкненого шляху об'єднує усі точки таким чином, щоб сума довжин ребр мала мінімальну сумарну довжину. У результаті його виконання будується граф, що в подальшому розрізається на кілька окремих графів по ребрах, довжина яких більше порогової. Алгоритм складається з набору простих дій:

1. Знайти пару точок з найменшою відстанню між ними та з’єднати їх ребром.
2. Знайти ізольовану точку, що знаходиться найближче до неізольованої.
3. З’єднати ці дві точки ребром.
4. Повторювати пункти 2 та 3 до тих пір, доки не залишиться ізольованих точок.
5. Видалити ребра, що перевищують поріг R (рис 1.18).

Недоліками цього алгоритму є те, що складно керувати кількістю кластерів, оскільки зручніше задавати кількість кластерів безпосередньо, ніж оперувати параметром максимальної допустимої довжини ребр результуючого графа. Крім того якщо точки слабко сконцентровані, то даний алгоритм не дає достатньо якісного результату. Найкращий результат даний алгоритм дає у випадку, коли точки сконцентровані у невеликі хмари або зібрані у фігури стрічкоподібної форми.

Алгоритм k-метроїдів - це алгоритм кластеризації, що пов'язаний з алгоритмом k-середніх та алгоритмом medoidshift. Обидва алгоритми k-середніх і k-метроідів є розділовими (розбивають набір даних на групи), і обидва намагаються мінімізувати відстань між точками, що належать кластерові, і точкою, позначеною як центр цього кластера.

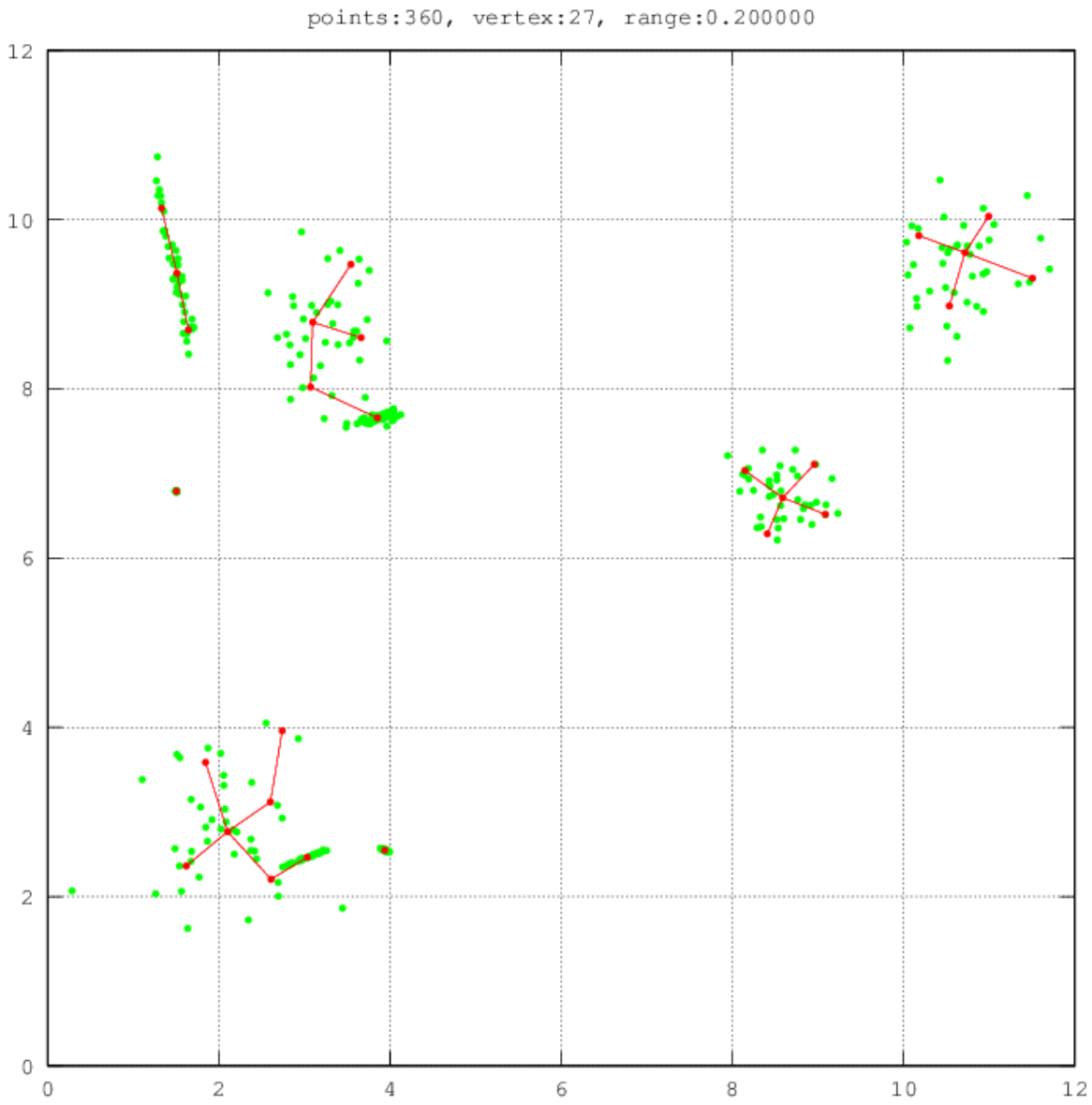


Рисунок 1.18. Результат після видалення найбільших ребер.

На відміну від алгоритму k-середніх, k-метроїдів вибирає точки даних як центри і працює з узагальненням мангеттенскої норми, щоб визначити відстань між даними точок.

k-метроїдів - це класична методика розбиття у кластеризації, яка кластеризує набір даних n об'єктів у k кластерів, відомих апріорно.

Він більш стійкий до шуму та викидів в порівнянні з k-середніх, оскільки він мінімізує суму попарних розбіжностей замість суми квадратів евклідової відстані.

Медоїд може бути визначений як об'єкт кластера, середня невідповідність якого всім об'єктам у кластері мінімальна. тобто він є найбільш централізованою точкою в кластері.

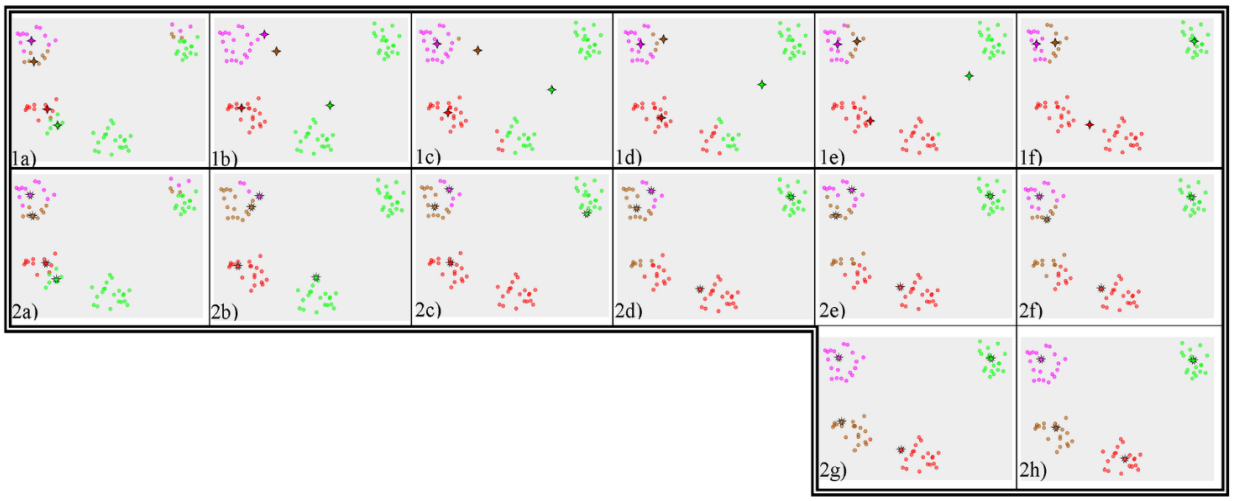


Рисунок 1.19. Порівняння k-середніх та k-метроїдів.

Як можна бачити з рис. 1.19, у даному випадку алгоритм k-середних (1a-1f) не зміг розпізнати очевидні кластери у наборі даних, в той час як алгоритм k-метроїдів(2a-2h) справився з цією задачею.

II. Чіткі та нечіткі.

Чіткі алгоритми кожному об'єкту вибірки ставлять у відповідність номер кластера [8], тобто кожен об'єкт відноситься тільки до одного кластера. Нечіткі алгоритми кожному об'єкту ставлять у відповідність набір значень, що показують ступінь належності об'єкта до кластерів [24, 25]. Тобто кожен об'єкт з певною мірою відноситься до кожного кластеру.

Найбільш популярним нечітким алгоритмом кластеризації є алгоритм Fuzzy C-Means (FCM) [26-29], що широко використовується для кластеризації різноманітних даних, починаючи від текстів і закінчуючи сегментацією зображень.

Алгоритм FCM мінімізує функціонал схожий на той, що мінімізується у k-середніх:

$$J = \sum_{k=1}^{M}\sum_{i=1}^{N} \mu_{ik}^{m} d(x_i, c_k),$$

де $x_i \in X$ - об'єкт кластеризації, $c_k \in C$ -центр мас кластера (центроїд), d - міра відстані, а μ_{ik}^{m}-коефіцієнт належності i-ї точки до k-го кластеру. Для даної точки даних x_i, ступінь її належності до кластера j розраховується наступним чином:

$$\mu_{ij} = \frac{1}{\sum_{k=1}^{N}\left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|}\right)^{\frac{2}{m-1}}}, \quad (1)$$

де m - коефіцієнт нечіткості, а вектор c_j розраховується наступним чином:

$$c_j = \frac{\sum_{i=1}^{M} \mu_{ij}^{m} x_i}{\sum_{i=1}^{M} \mu_{ij}^{m}}, \quad (2)$$

У рівняннях (1) і (2) коефіцієнт нечіткості m, де 1 <m<∞, визначає толерантність кластеризація. Це значення визначає, скільки кластерів може перетинатися один з одним. Чим вище значення m, тим більше кластери перекривають один одного. Іншими словами, чим більший коефіцієнт

нечіткості алгоритм використовує, тим більше число точок даних буде потрапляти до смуги де ступінь належності до кластера є не нулем або одиницею, а знаходиться у проміжку між ними.

Приклад кластеризації алгоритмом FCM зображено на рис. 1.20.

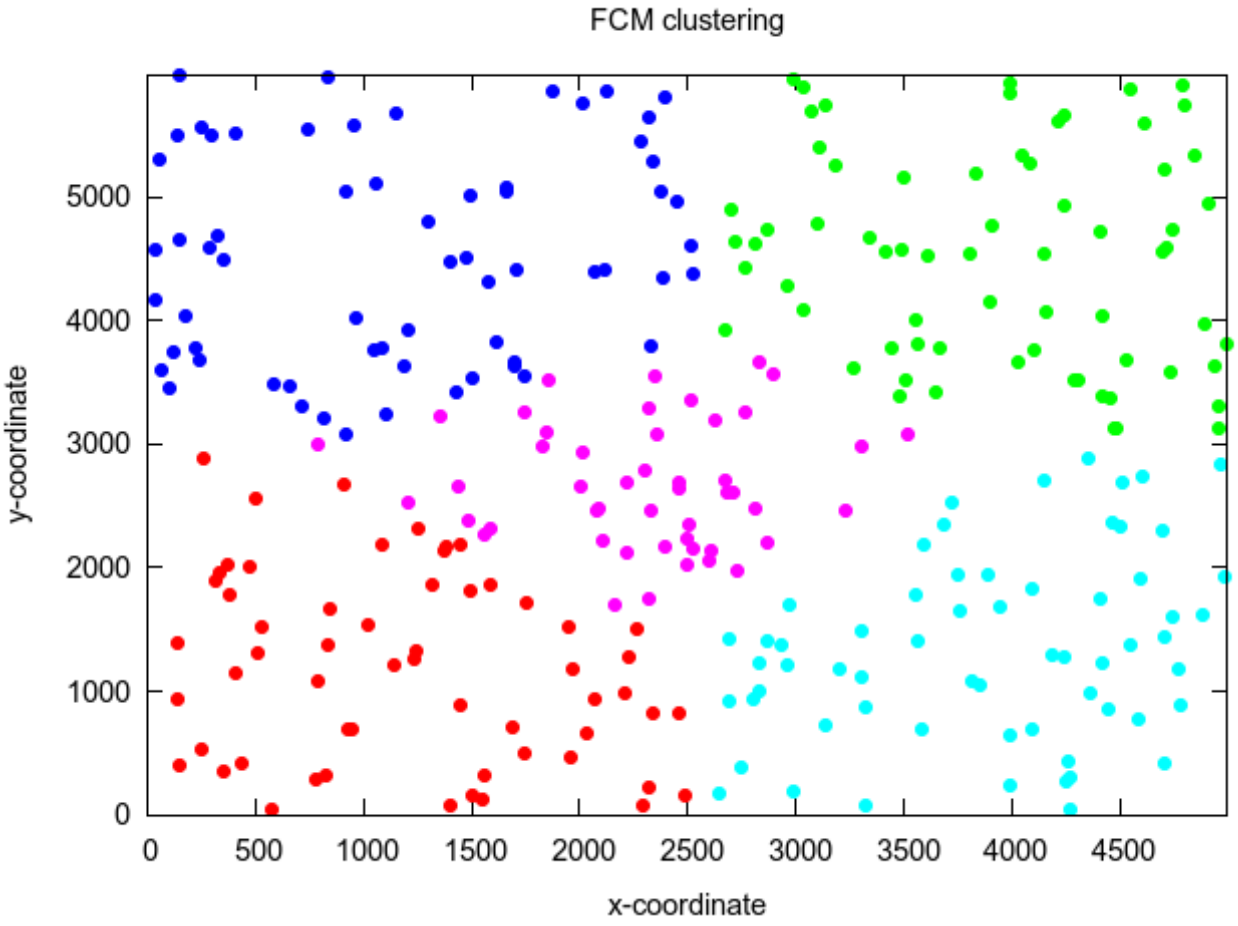


Рисунок 1.20. Результат кластеризації алгоритмом FCM.

Загалом алгоритм FCM працює подібно до алгоритму k-середніх, з використанням нечіткої логіки. Тому для цього алгоритму також є характерною проблема кількості кластерів. Крім того з’являється проблема вибору оптимального коефіцієнта нечіткості.

Висновки до розділу 1

У даному розділі було розглянуто кластеризацію, як задачу інтелектуального аналізу даних. Було розглянуто сучасні алгоритми кластеризації, їх класифікацію та наведено приклади результатів їх роботи. Висвітлено можливі проблеми та недоліки цих алгоритмів.

РОЗДІЛ 2
МАТЕРІАЛИ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1. Мова програмування Java

Java є мовою програмування і платформою обчислень, яка була вперше випущена Sun Microsystems в 1995 р [30]. Програми Java компілюються у спеціальний байт-код, який виконується на спеціальному інтерпретаторі – віртуальній машині Java (JVM) (рис 2.1).

Перевагою подібного способу виконання програм є повна незалежність байт-коду від операційної системи і устаткування, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина. Іншою важливою особливістю технології Java є гнучка система безпеки, в рамках якої виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), викликають негайне переривання.

Основні можливості Java:

- автоматичне керування пам'яттю;
- розширені можливості обробки виняткових ситуацій;
- багатий набір засобів фільтрації введення-виведення;
- набір стандартних колекцій: масив, список, стек і т. п.;
- наявність простих засобів створення мережевих додатків (у тому числі з використанням протоколу RMI);
- наявність класів, що дозволяють виконувати HTTP-запити і обробляти відповіді;

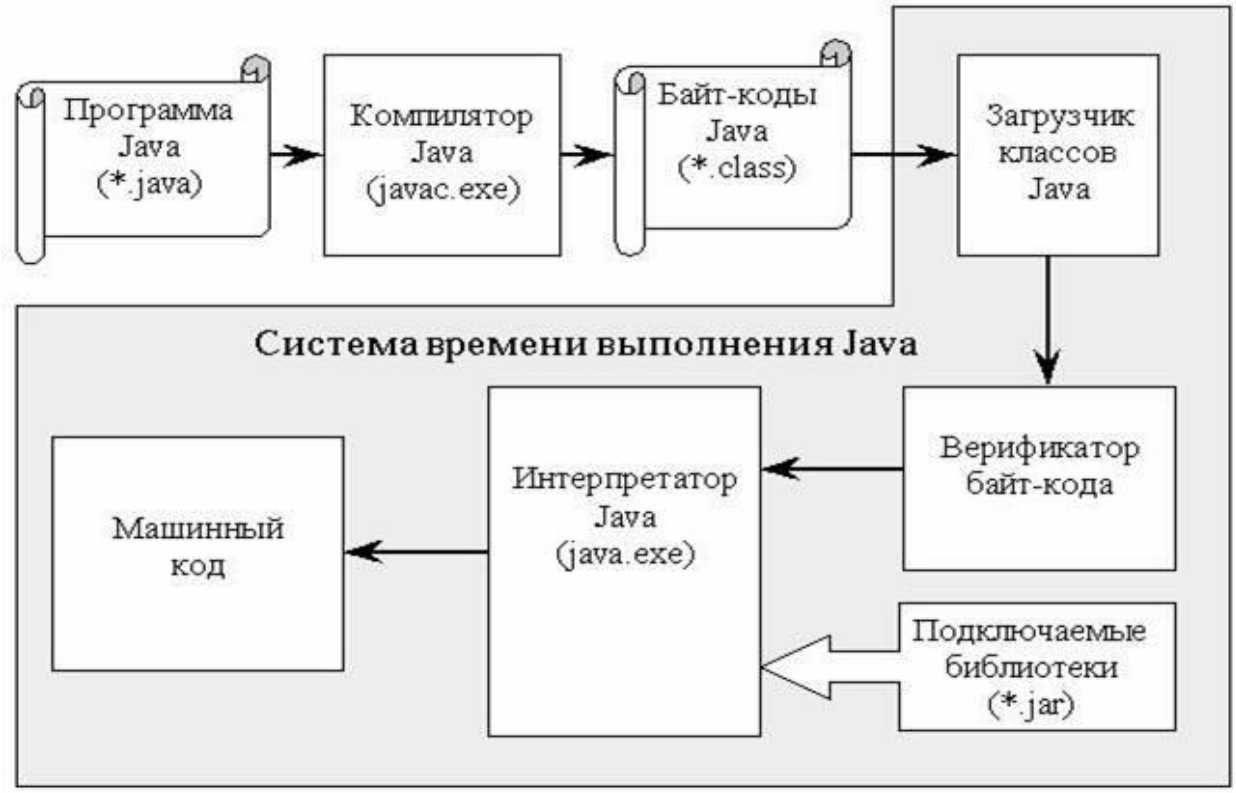


Рисунок 2.1. Виконання програми на Java.

- вбудовані в мову засоби створення багатопоточних додатків;
- уніфікований доступ до баз даних, як на рівні окремих SQL-запитів - на основі JDBC, так і на рівні концепції об'єктів, що мають здатністю до зберігання в базі даних - на основі Java Data Objects і Java Persistence API;
- підтримка шаблонів (generics);
- підтримка лямбда функцій, замикань, вбудовані можливості функціонального програмування;

Java застосовується у різноманітних сферах:

- для написання програмних продуктів для стаціонарних ПК та ноутбуків без підтримки мережевих технологій;
- для написання програмних додатків для мобільних пристроїв та телефонів;

- для програмування плат як промислового, так і побутового значення;

Основною мовою програмування обрано Java, оскільки вона:

- Є об’єктно-орієнтованою мовою програмування, що дає можливість багаторазово використовувати вже написані модулі в інших проектах.
- Є кросс-платформною мовою програмування, що не обмежує можливості розробника та не прив’язує його до конкретної платформи.
- Має добре організовану систему обробки помилок, а також спрощену роботу з пам’яттю.
- Має простий та зручний механізм роботи з потоками.
- Має багато бібліотек з відкритим вихідним кодом.

2.2. Мова програмування Scala

Мова програмування Scala поєднує в собі як функціональних, так і об’єктно-орієнтований підходи. Scala - це чисто об'єктно-орієнтована мова в тому сенсі, що кожне значення є об'єктом [31]. Типи та поведінка об'єктів описуються класами та рисами. Класи поширюються на підкласування та гнучкий механізм композиції на основі суміші, що є чистою заміною для багаторазового спадкування.

Scala також є функціональною мовою в тому сенсі, що кожна функція є значенням. Scala забезпечує простий синтаксис для визначення анонімних функцій, підтримує функції вищого порядку, дозволяє виконувати функції вкладання та підтримує каррінг. Архітектура класів Scala та його вбудована підтримка алгебраїчних типів моделі зіставлення з шаблоном, що використовуються в багатьох мовах функціонального програмування. Об'єкти-синглтони забезпечують зручний спосіб об'єднання функцій, які не є членами класу.

Класи стандартної бібліотеки Scala надають широкі можливості для обробки даних, що дуже корисно у контексті поставлених задач. Крім того Scala призначена для взаємодії з популярним Java Runtime Environment (JRE). Зокрема, взаємодія з об'єктно-орієнтованою мовою програмування Java настільки гладка, наскільки це можливо. Нові функції Java, такі як лямбда функції, анотації та шаблони (generics), мають прямі аналоги в Scala.

Ті можливості Scala, що не мають аналогів у Java, такі як значення за замовчуванням та іменовані параметри, компілюються так близько до Java, наскільки це можливо. Scala має таку ж модель компіляції, як Java, і дозволяє отримати доступ до тисяч існуючих високоякісних бібліотек.

Саме тому мова програмування Scala була вибрана як додаткова для реалізації окремих модулів обробки даних.

2.3. Платформа JavaFX

JavaFX — є платформою та інструментарієм для створення інтернет-застосунків з можливістю підвантаження медіа та змісту. JavaFX включає в себе набір утиліт, для швидкого створення розвинутих інтернет-застосунків для різноманітних платформ.

Платформа JavaFX 2.x включає в себе наступні компоненти:

- Інструмент JavaFX SDK: інструменти виконання. Графіка, медіа-веб-сервіси та текстові бібліотеки. У Java FX 1.x також включений компілятор JavaFX, який зараз застарілий, оскільки код користувача JavaFX написано на Java.
- NetBeans IDE для JavaFX: NetBeans з палітрою перетягування, щоб додати об'єкти з перетвореннями, ефектами та анімацією, а також набір зразків та найкращих практик.
- Сценарій JavaFX, що було представлено для Java FX 2.1 і вище. Інтерфейс користувача створюється шляхом перетягування елементів

керування з палітри. Ця інформація зберігається як файл FXML, спеціальний XML-формат (рис 2.2).

- Інструменти та плагіни для творчих інструментів (aka Production Suite): плагіни для Adobe Photoshop та Adobe Illustrator, які можуть експортувати графічні об'єкти в код сценарію JavaFX, інструменти для перетворення SVG-графіки у код JavaFX Script та попередній перегляд активів, конвертованих у JavaFX з інших інструментів (в даний час не підтримується в версіях JavaFX 2.x).

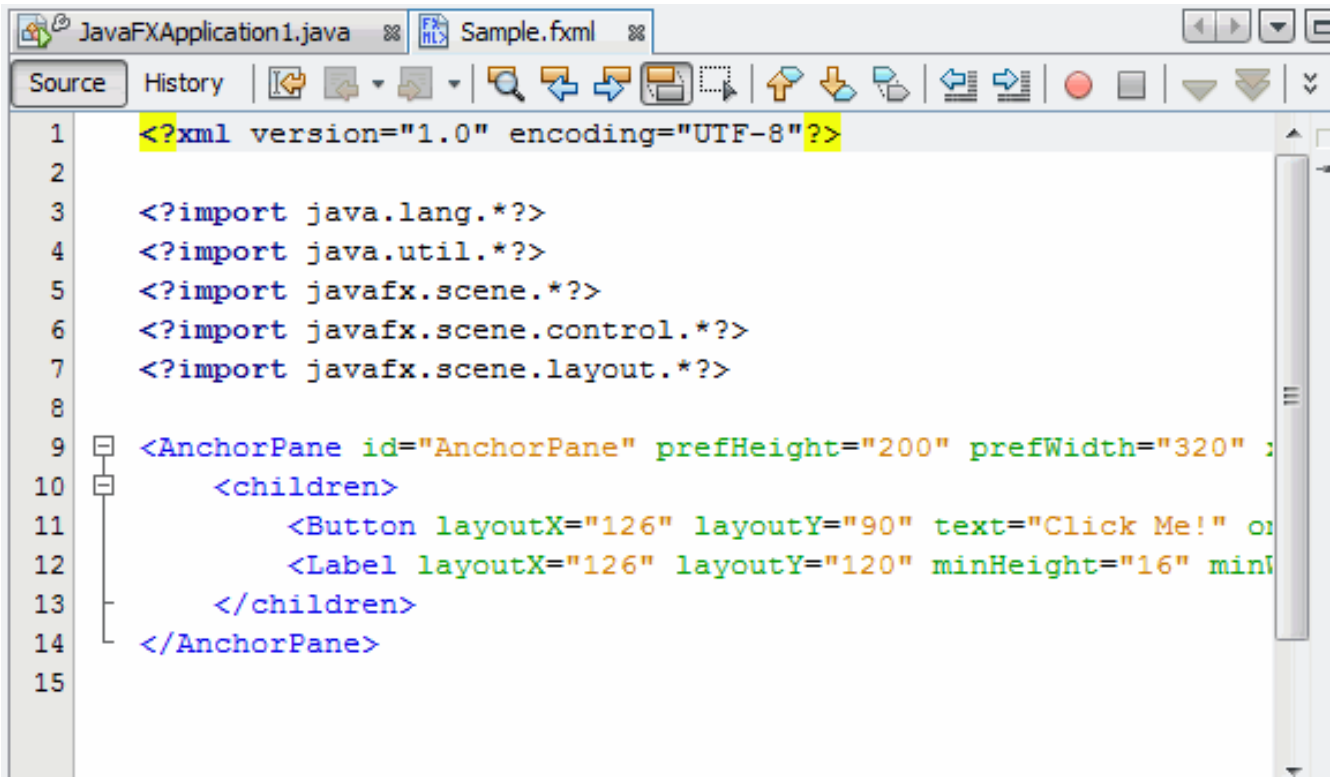


Рисунок 2.2. Приклад файлу FXML.

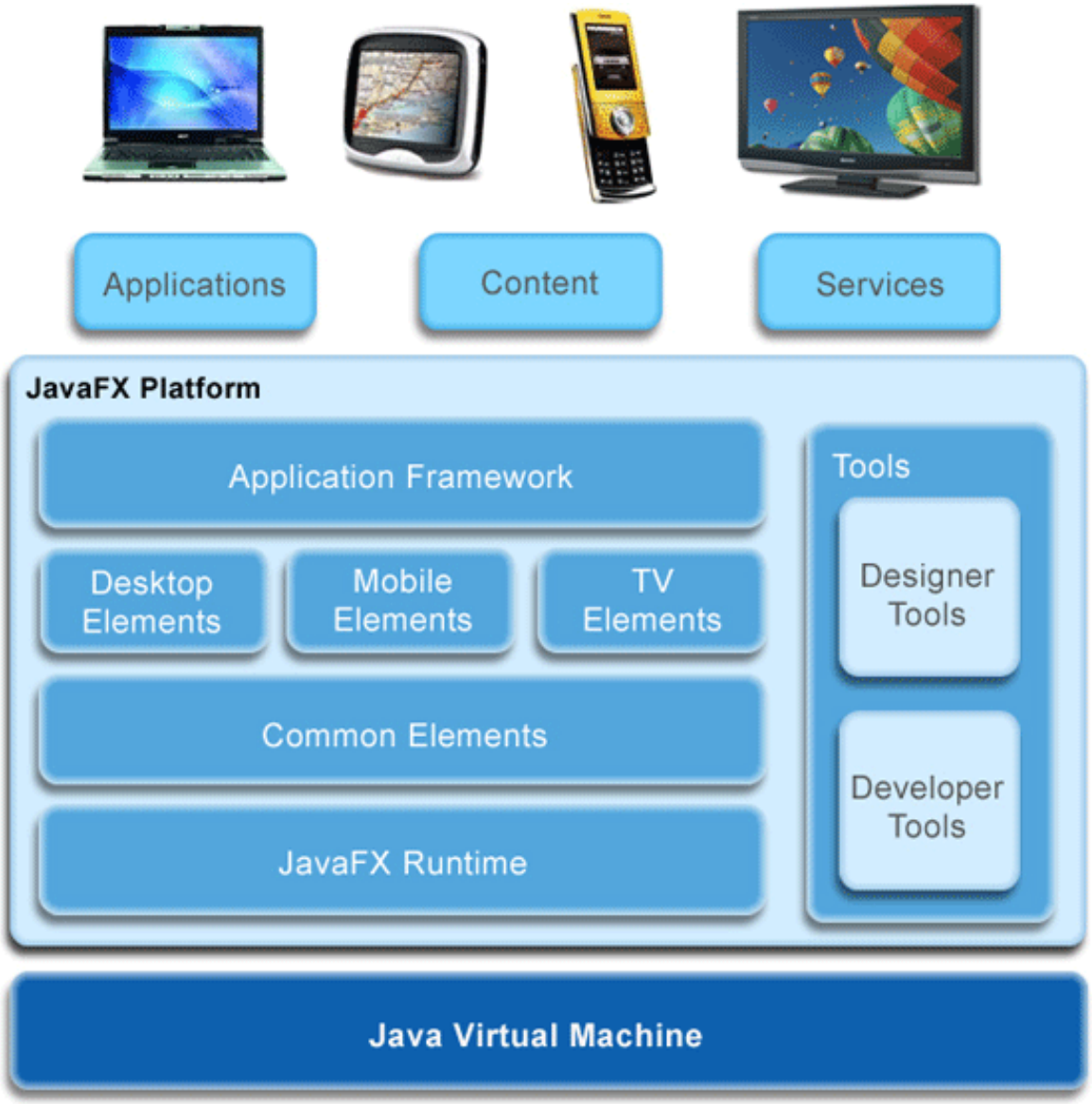


Рисунок 2.3. Архітектура платформи JavaFX.

На рис. 2.3 зображено архітектуру платформи JavaFX, що включаэ в себе середу ваконання, загальні елементі та програмний фреймворк. Крім того платформа включає в себе інструменти для розробки та дизайну. JavaFX Scene Builder надає можливість розробляти інтерфейс програмного додатку інтерактивно. Він існує як самостійне ПЗ (рис 2.4) та як частина інтегрованого середовища розробки.

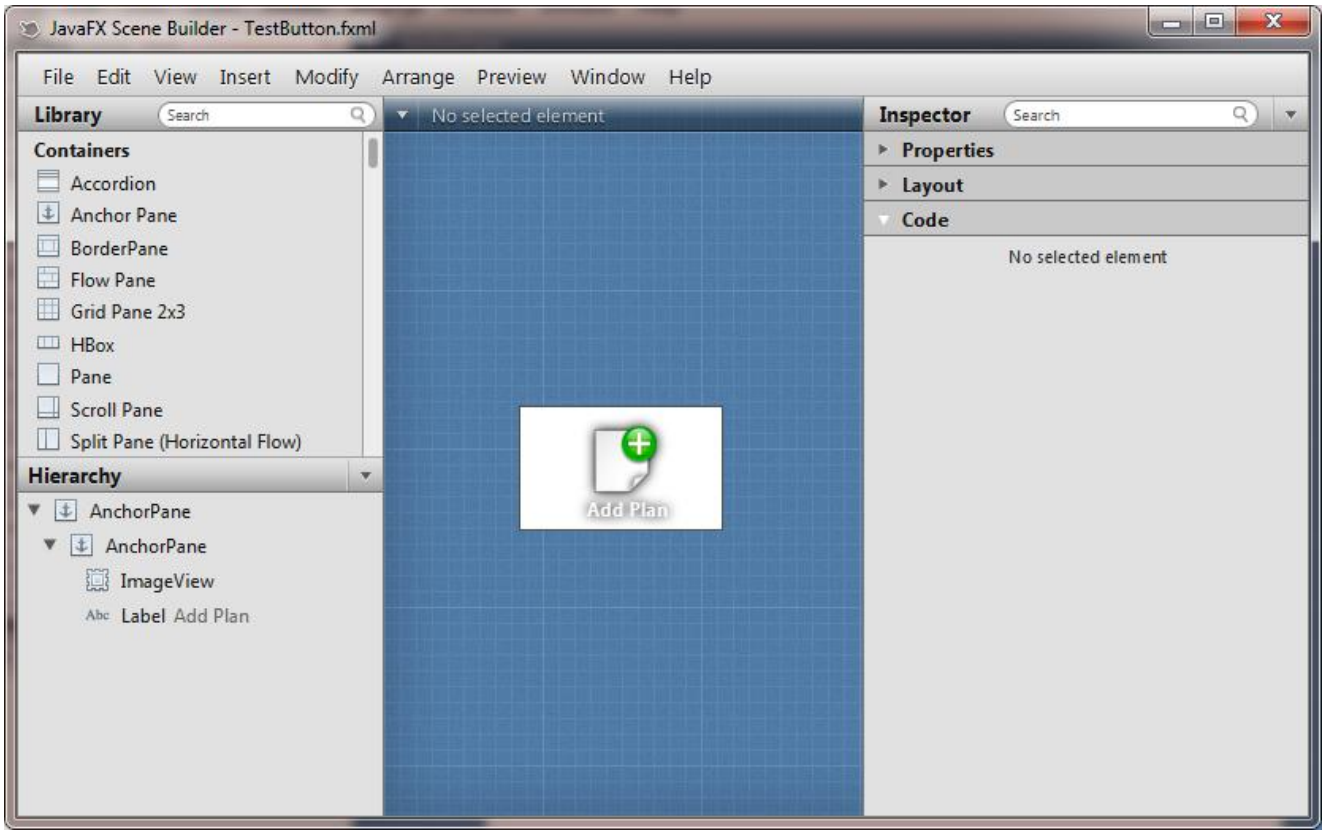


Рисунок 2.4. JavaFX Scene Builder.

2.4. Бібліотека java swing

Swing — інструментарій для створення графічного інтерфейсу користувача (GUI) мовою програмування Java [32]. Це частина бібліотеки базових класів Java (*JFC*, Java Foundation Classes).

Swing було розроблено для забезпечення більш функціонального набору програмних компонентів для створення графічного інтерфейсу користувача, ніж у існуючого на той час інструментарію AWT. Компоненти цієї бібліотеки підтримують специфічні look-and-feel модулі, що підключаються динамічно. Вони дозволяють емуляцію графічного інтерфейсу платформи (динамічне підключеннння специфічних для певної платформи вигляду та поведінки). Основним недоліком таких компонентів є відносно повільна робота. Позитивною стороною є універсальність

інтерфейсу створених програм на всіх платформах. Приклад Swing інтерфейсу зображено на рис 2.5.

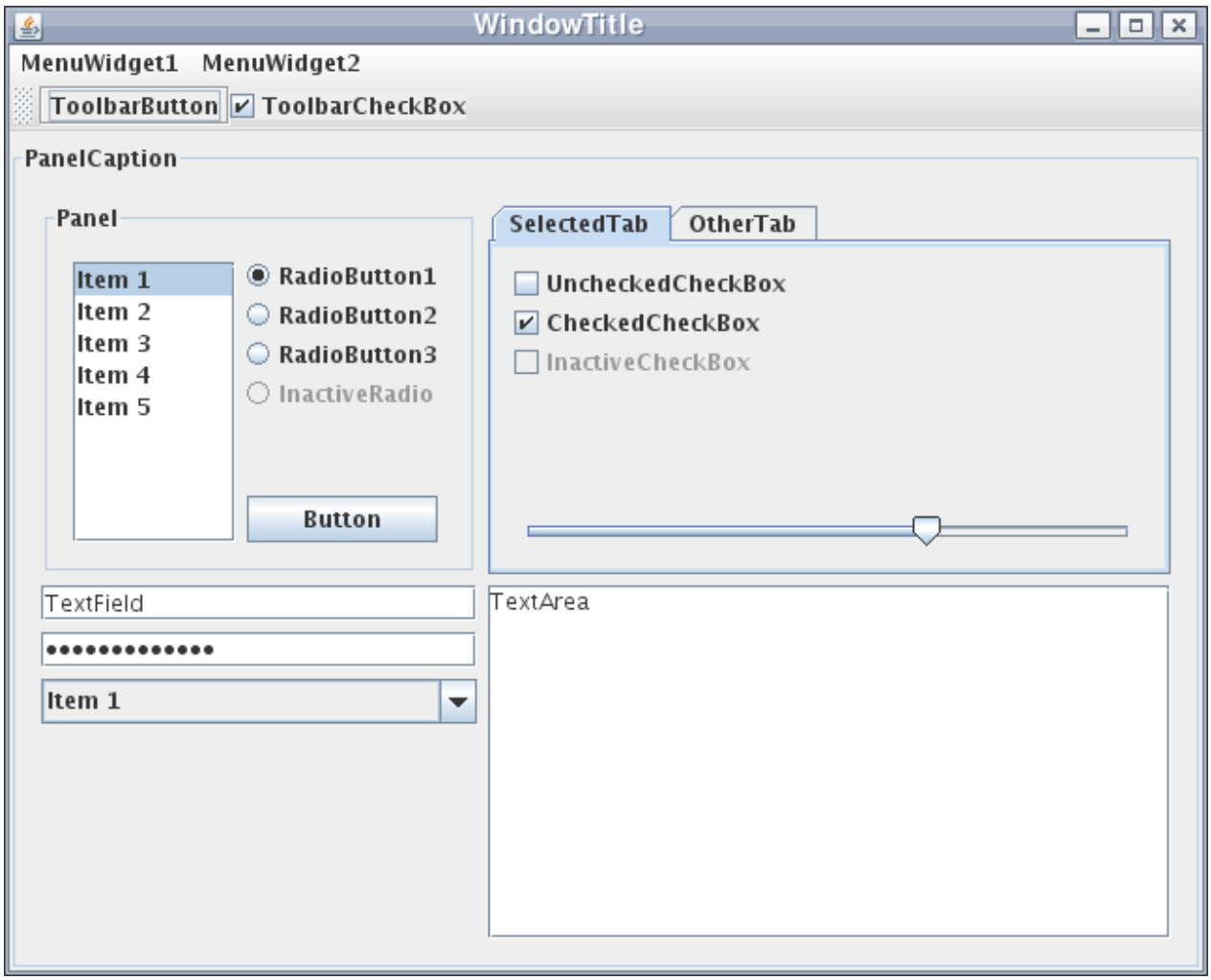


Рисунок 2.5. Інтерфейс Swing.

2.5. F1-міра

У статистичному аналізі бінарної класифікації оцінка F1 (також F-score або F-measure) є мірою точності тесту [33]. Вона розглядає як точність (Precission) P, так і специфічність R (Recall) тесту для обчислення оцінки: p - кількість правильних позитивних результатів, поділених на кількість всіх позитивних результатів, що повертаються класифікатором, K - кількість правильних позитивних результатів, поділених на кількість всіх відповідних зразків (всі зразки, які повинні були бути ідентифіковані як позитивні).

Оцінка F1 - це гармонійне середнє значення точності і відкликання, де оцінка F1 досягає найкращого значення в 1 (ідеальні точність і чутливість), а найгірша - на 0.

У випадку коли точність та специфічність мають однаковий пріоритет, то F-міра має виглад зображений на рис. 2.6.

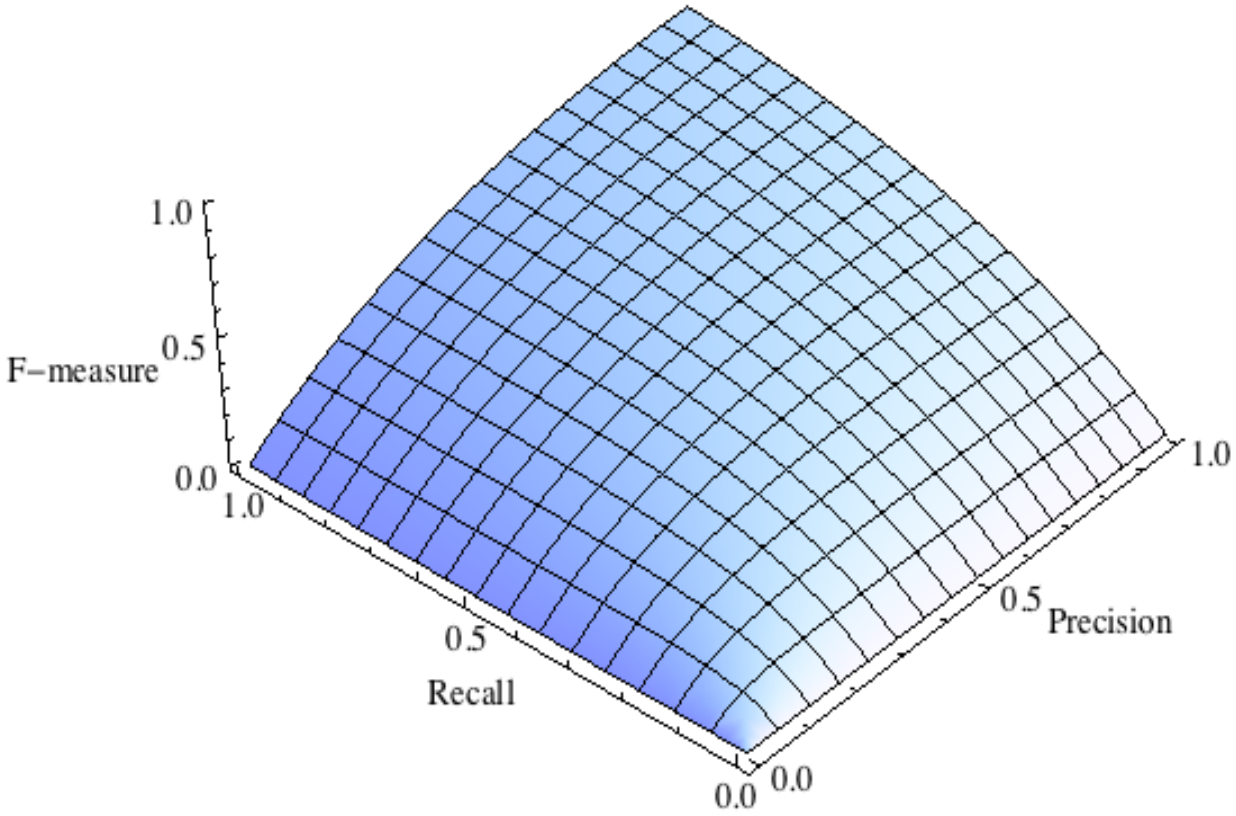


Рисунок 2.6. F_1-міра.

F-міра часто використовується в області пошуку інформації для вимірювання пошуку, класифікації документів та характеристик класифікації запитів. Раніше роботи зосереджувалися головним чином на оцінці F_1, але при поширенні великомасштабних пошукових систем цілі продуктивності змінювалися, щоб більше уваги робити на точність або специфічність.

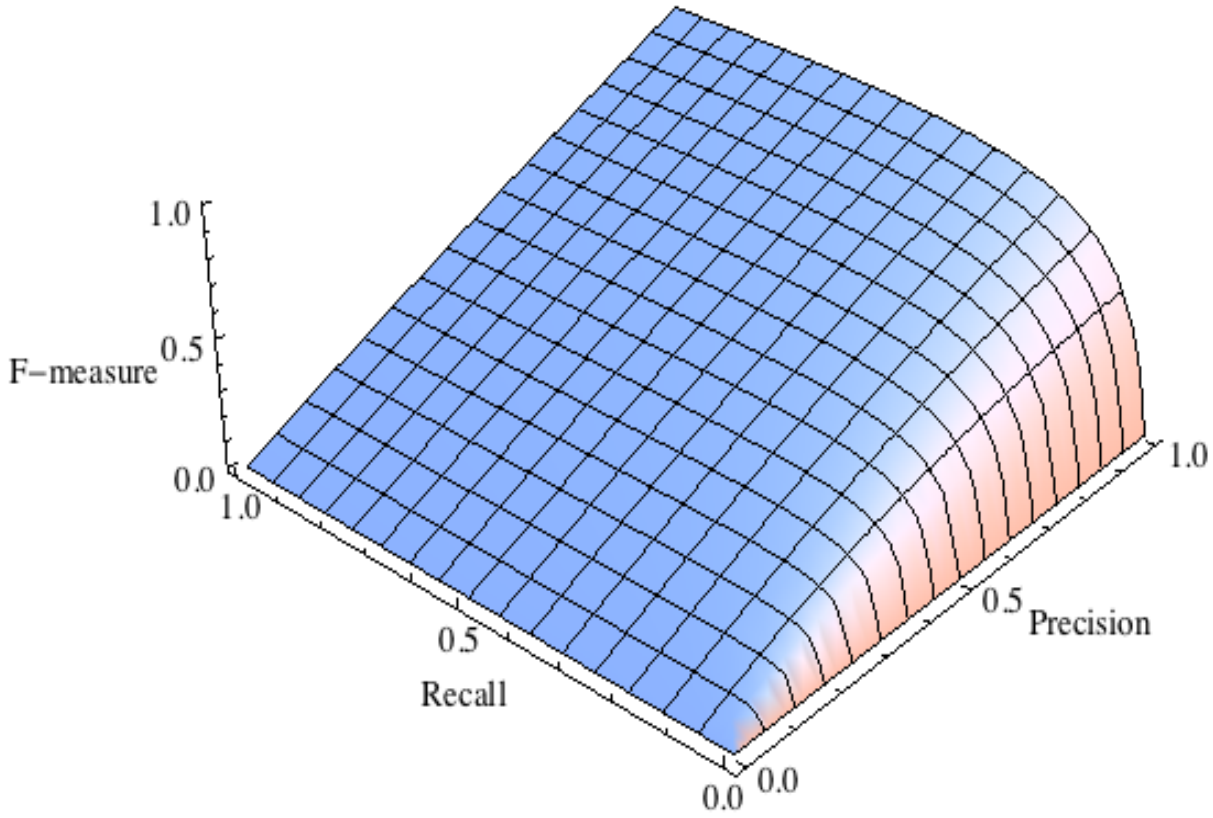


Рисунок 2.7. Пріоритет точності.

У загальному випадку формула F-міри буде мати наступний вигляд:

$$F = (\beta^2 + 1)\frac{\mathrm{Pr}\,ecission \cdot \mathrm{Re}\,call}{\beta^2\,\mathrm{Pr}\,ecission + \mathrm{Re}\,call}$$

У разі пріоритету точності F-міра буде мати вигляд зображений на рис. 2.7, якщо ж пріоритет віддається специфічності, то вона буде виглядати як на рис. 2.8.

Перевагою даного методу оцінки є його відносна простота у розрахунку. Недоліком ж буде те, що його використання перевірки якості кластеризації не завжди можливе, оскільки він потребує наявності вже розділених на групи даних.

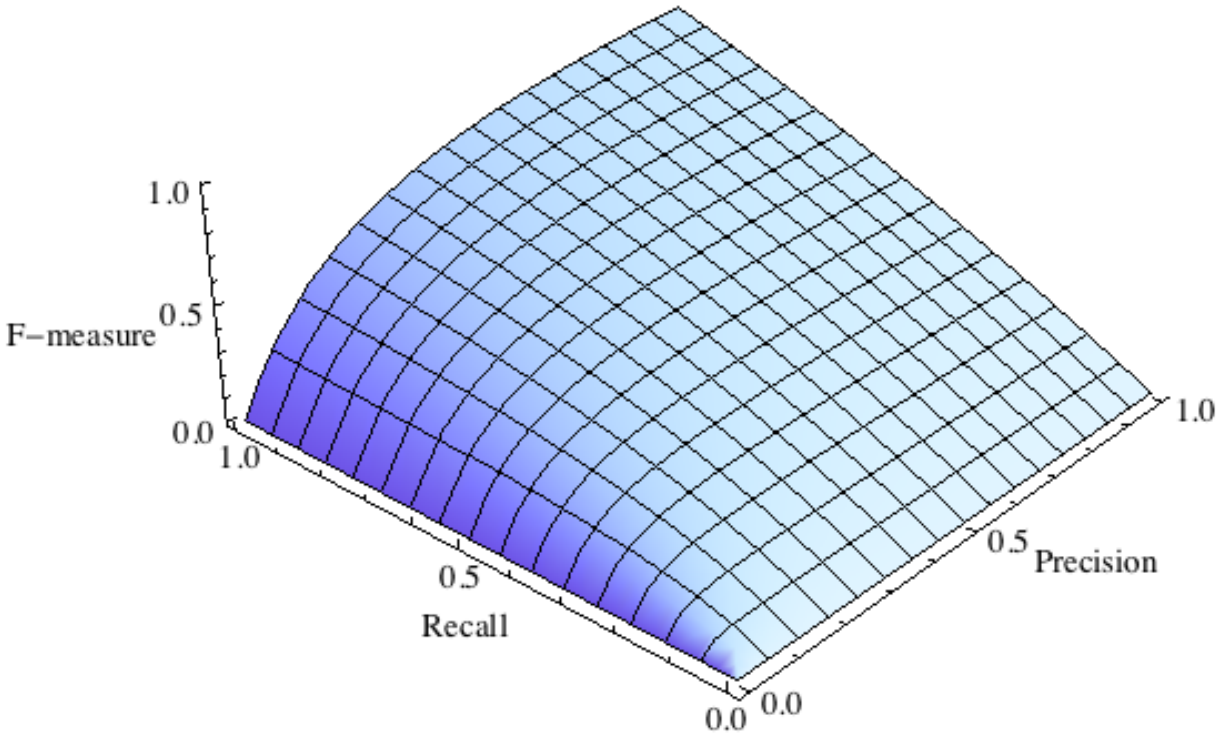


Рисунок 2.8. Пріоритет специфічності.

2.6. Apache POI

Apache POI – це проект, яким керує фонд Apache Software і раніше підпроект проекту Jakarta. Він надає чисті бібліотеки Java для читання та запису файлів у форматах Microsoft Office, таких як Word, PowerPoint та Excel.

Проект Apache POI містить наступні підкомпоненти [34]:

- POIFS (Poor Obfuscation Implementation File System) - цей компонент зчитує та записує формат документів Microsoft OLE 2 Compound. Оскільки всі файли Microsoft Office є файлами OLE 2, цей компонент є основним будівельним блоком всіх інших елементів POI. POIFS, таким чином, може використовуватися для читання більшого числа файлів, крім тих, чиї явні декодери вже написані в POI.

- HSSF (Horrible SpreadSheet Format) - читає та записує файли формату Microsoft Excel (XLS). Він може читати файли, написані Excel 97 і надалі; цей формат файлів відомий як формат BIFF 8. Оскільки формат файлу Excel складний і містить ряд складних характеристик, деякі з додаткових функцій не можуть бути прочитані.
- XSSF (XML Spread Sheet Format) - читає та записує файли формату Office Open XML (XLSX). Подібна функція встановлена для HSSF, але для файлів Office Open XML.
- HPSF (Horrible Property Set Format) - читає інформацію про "документі" з файлів Microsoft Office. Це, по суті, інформація, яку можна побачити, використовуючи пункт меню Файл | Властивості в програмі Office.
- HWPF (Horrible Word Processor Format) - призначений для читання та запису файлів формату Microsoft Word 97 (DOC). Цей компонент знаходиться на початкових етапах розробки.
- XWPF (XML Word Processor Format) - подібна функція встановлена для HWPF, але для файлів Office Open XML.
- HSLF (Horrible Slide Layout Format) - це чиста реалізація Java для файлів Microsoft PowerPoint. Це дає змогу читати, створювати та редагувати презентації (хоча деякі речі легше робити, ніж інші).
- HDGF (The Horrible DiaGram Format) - початкова чиста реалізація Java для бінарних файлів Microsoft Visio. Він забезпечує можливість читання вмісту низьких рівнів файлів.
- HPBF (The Horrible PuBlisher Format) - це чиста реалізація Java для файлів Publisher у Microsoft.
- HSMF (Horrible Stupid Mail Format) - це чиста реалізація Java для файлів MSG MS Outlook.
- DDF (Dreadful Drawing Format) - це пакет для декодування формату Microsoft Office Drawing.

Було використано класи XSSF для роботи з файлами створеними у Excel 2007 та вище. Причиною для вибору саме цієї бібліотеки послугувала її висока популярність та те, що її розробка керується професіоналами з фонду Apache Software.

2.7. Коефіцієнт силуета

Коефіцієнт силуета [35] є методом інтерпретації та валідації у кластерних даних. Значення силуета розраховується окремо для кожної точки (рис. 2.9), що належить кластеру і дозволяє оцінити як якість побудованого розбиття, так і кількість кластерів у самій вибірці.

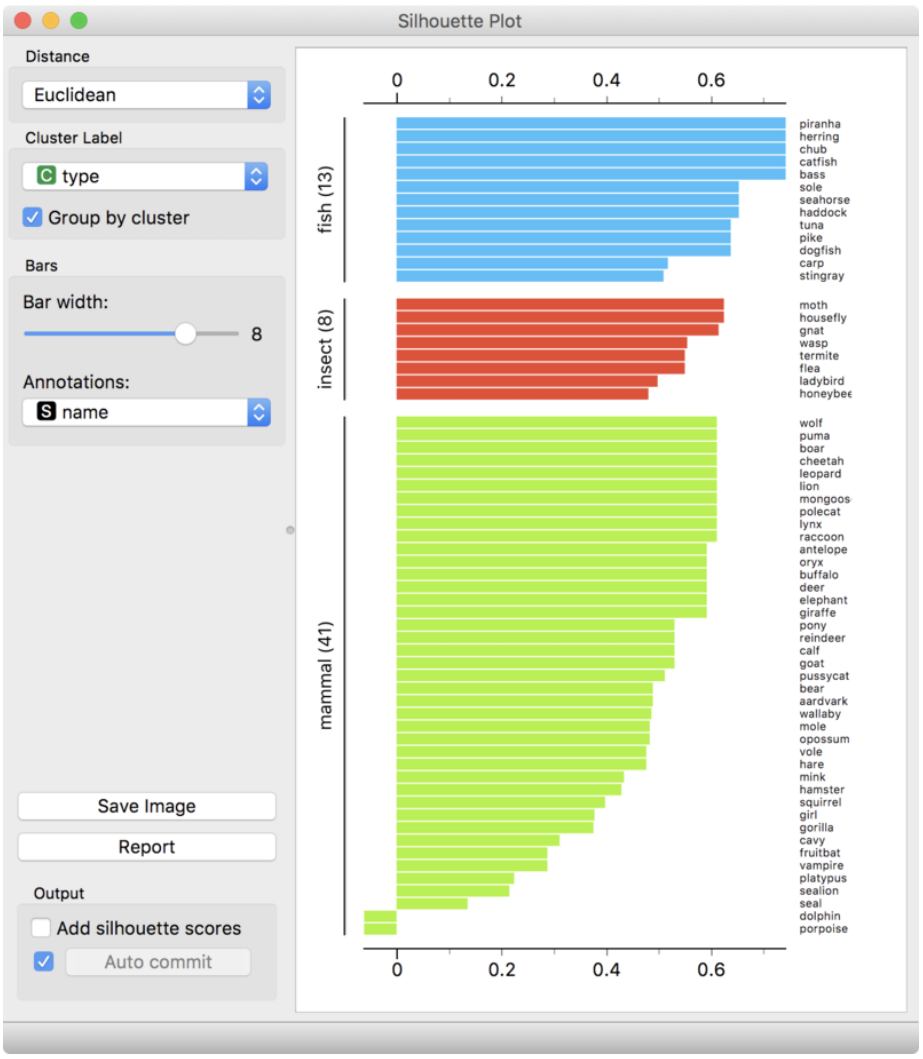


Рисунок 2.9. Графік коефіцієнта силуета по кластерах.

Коефіцієнт силуета показує, наскільки об’єкт близький до власного кластера по відношенню до інших кластерів. Коефіцієнт приймає значення від -1 до 1. Чим більше значення коефіцієнта, тим більше об’єкт підходить до власного кластера, та меньше до усіх інших. Якщо у більшості об’єктів високе значення коефіцієнта, то отримане розбиття є валідним. Якщо ж у багатьох точок значення коєфіцієнта є низьким, то це свідчить про надмірну (рис 2.10) або недостатню кількість кластерів (рис 2.9). Найкраще розбиття досягається коли відстань всередині кластера є малою, а відстань до елементів сусідніх кластерів є великою.

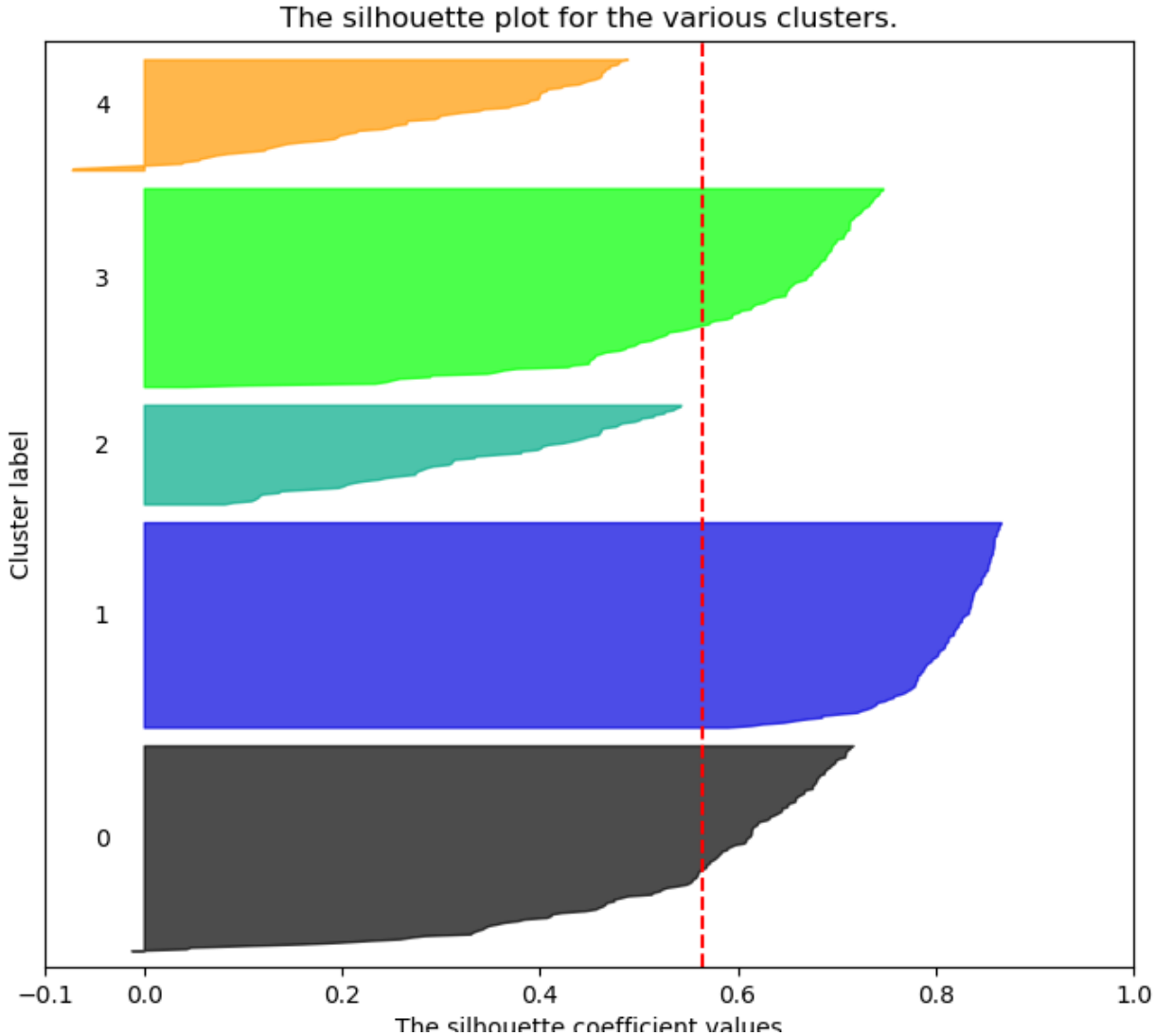


Рисунок 2.10. Коефіцієнт силуета при надмірній кількості кластерів.

Як можна бачити, якщо задано надмірну кількість кластерів, то з’являються кластери з коєфіцієнтом силуета, що є нижчим за середнє значення для розбиття. У випадку ж недостатньої кількості кластерів або наявності викидів коефіцієнт окремих точок всередині кластера буде набагато нижчий, ніж у інших членів кластера.

Коефіцієнт, у загальному випадку, розраховується наступним чином:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \qquad (2.1)$$

де *b(i)* – середня відстань від *i*-тої точки до усіх точок найближчого до неї кластера (крім того, до якого вона належить), *a(i)* – середня выдстань до всіх інших точок, що належать до того ж кластера.

Формулу (2.1) також можна представити у вигляді системи:

$$s(i) = \begin{cases} 1 - \dfrac{a(i)}{b(i)}, a(i) < b(i) \\ 0, \quad a(i) = b(i) \\ \dfrac{b(i)}{a(i)} - 1, a(i) > b(i) \end{cases}$$

Для розрахунку індекса силуета для певного кластера або розбиття в цілому, розраховують середнє арифметичне. На практиці також використовуються інші варіанти індекса силуета: спрощений силует та альтернативний силует [36].

Спрощений силует використовує замість відстані між одним елементом та усіма іншими відстань від елемента до відповідних центроїдів, що підходить для алгоритмів, що формують кластери гіперсферичної форми, наприклад k-середніх.

Альтернативний індекс силуета виглядає наступним чином:

$$s(i) = \frac{b(i)}{a(i) + \varepsilon},$$

де ε – достатньо мала константа, яку було введено щоб уникнути ділення на нуль.

2.8. Середовище розробки IntelliJ IDEA

IntelliJ IDEA — це комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala та інші) від компанії JetBrains. Інтерфейс Intellij IDEA дозволяє викликати більшість функцій IDE не покидаючи вікна редактора.

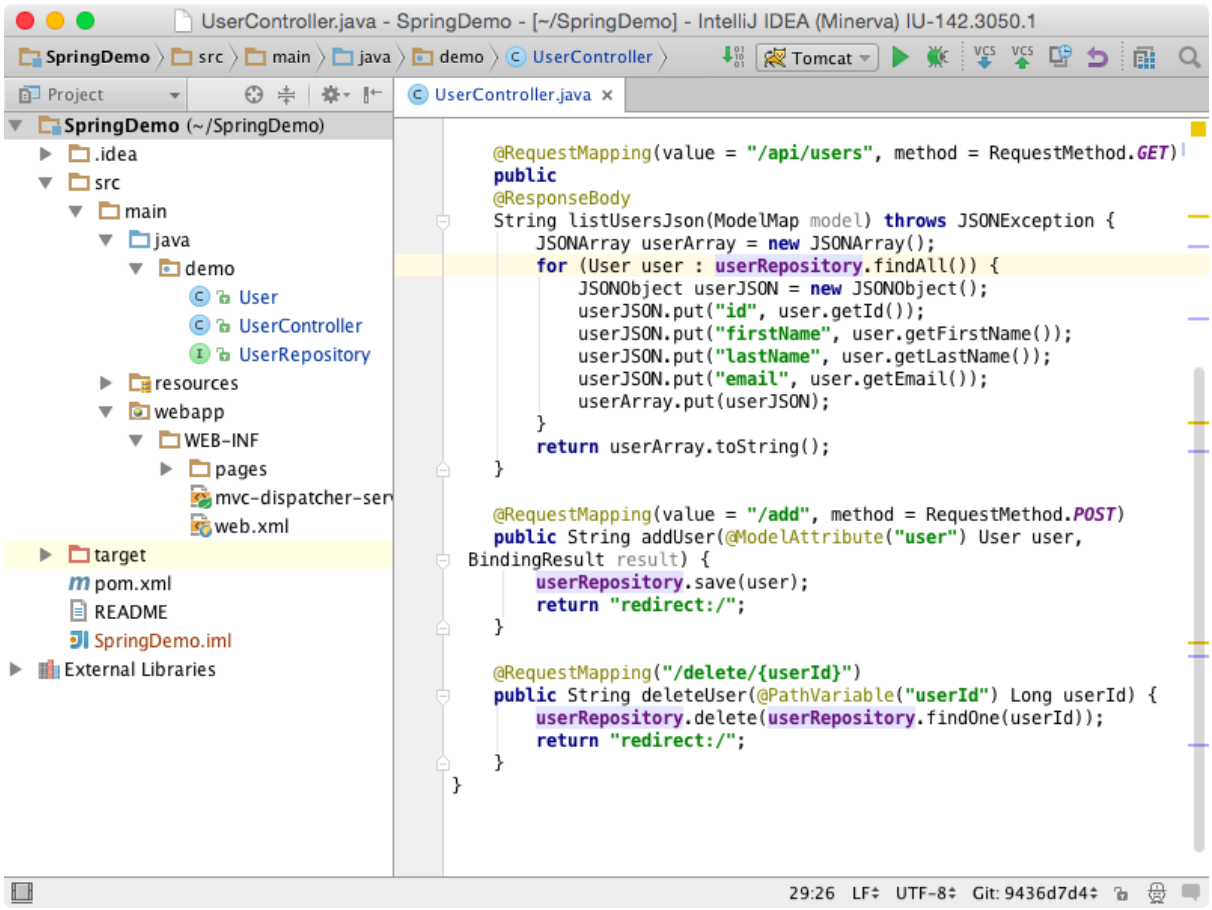


Рисунок 2.11. Інтерфейс IntelliJ IDEA.

На рис. 2.11 представлений базовий вид середовища розробки. Як видно з рисунка, область зліва містить менеджер файлів проекту для швидкого доступу та зручної навігації. Блок справа є основним і містить вкладки з відкритими для редагування файлами. Слід також зазначити, що подібне розміщення вікон є за замовчуванням, але користувач може у досить широкому об’ємі регулювати їх положення та розміри.

Середовище розробки підтримує встановлення різноманітних плагінів (рис. 2.12) для додання нових можливостей, таких як підтримка нових мов програмування, засоби для роботи з певними форматами файлів, засоби для форматування коду.

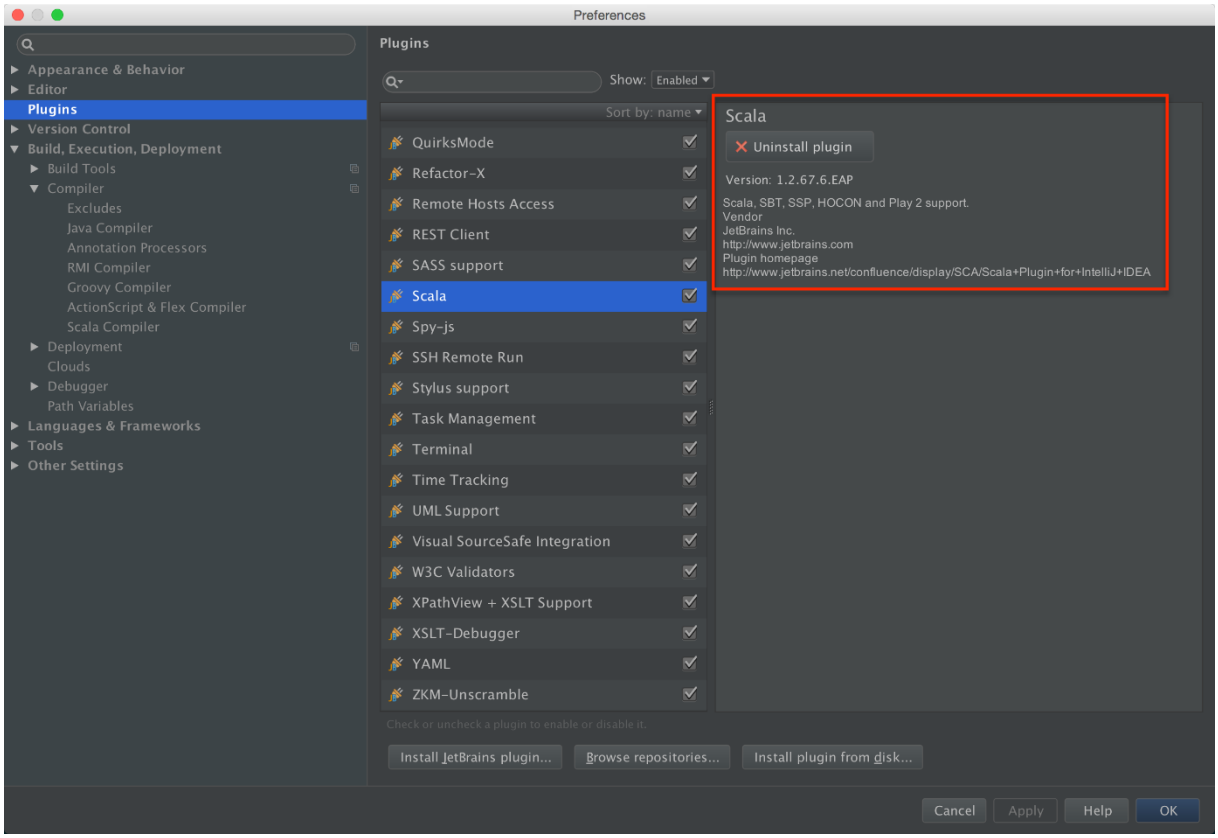


Рисунок 2.12. Інтерфейс IntelliJ IDEA.

Середовище розробки підтримує засоби автоматичної збірки, такі як Maven та SBT.

Стосовно безпосередньо написання коду, підтримується багато мов програмування, для яких реалізовано автодоповнення коду та засоби аналізу його якості.

Дане середовище розробки було вибрано тому, що воно надає широкі можливості для роботи з програмами, що використовують декілька мов.

Висновки до розділу 2

У даному розділі було описано використані у дослідженні матеріали та методи. Було наведено причини, що послугували для їх вибору. Крім того, було зазначено їх переваги та недоліки.

РОЗДІЛ 3
РОЗРОБКА НЕЧІТКОГО АЛГОРИТМУ К-СЕРЕДНІХ З ОБМЕЖЕНОЮ МАСОЮ РОБОЧОЇ ОБЛАСТІ

3.1. Математичний опис алгоритму

В якості вихідного був обраний метод кластеризації К-середніх з обмеженою масою кластера [37]. Суть даного методу полягає в тому, що в процесі формування кластерів не перевищується задана кількість I_{max} об'єктів в кластері, тобто обмежується маса та інерція кластера шляхом "забування" накопиченої раніше інформації. При приєднанні l-го об'єкта до деякого кластеру c_t, що містить в собі n_t об'єктів розрахунок нового положення центроїда x_{ct} здійснюється за формулами:

$$n_t < I_{\max} \Rightarrow x_{c_t} = \frac{\sum_{i=1}^{n_t} x_i + x_l}{n_t}, \tag{3.1}$$

$$n_t = I_{\max} \Rightarrow x_{c_t} = \frac{\sum_{i=1}^{n_t} x_i + x_l - p \cdot m}{I_{\max} - p}, n_t = n_t - p. \tag{3.2}$$

У формулах (3.1) та (3.2) сума $\sum_{i=1}^{I_{max}} x_i + x_l$ являє собою накопичену суму приєднаних об’єктів, p – кількість об'єктів, що забуваються, m – середнє значення наявних в кластері об'єктів.

Даний алгоритм було розроблено для вирішення задачі пошуку трендів у просторі зниженої розмірності.

У ході роботи вихідний алгоритм було модифіковано для використання функції належності [38, 39].

Послідовність дій у алгоритмі наступна:

1. Проводиться нормування даних.

2. Ініціалізація центрів кластерів, яка відбувається одним із таких способів:

- Найближчі до початку координат.
- Найближчі до центру мас множини точок в просторі ознак.
- На периферії множини точок в просторі ознак.
- Рівномірно віддалені від центру з заданим кроком.
- Найбільш віддалені від початку координат.
- Вибрані з окремих міркувань.
- Вибрані випадковим чином.

3. Задається рівень інерційності кластерів і кількість об’єктів, що виключаються з розрахунку нового положення центроїдів.

4. До початкових центрів кластерів приєднуються найближчі об'єкти.

5. Розраховується значення міри належності для кожної точки.

6. Процедура завершується після обходу усіх точок, якщо не задано додаткових умов.

Очевидно, що описаний вище алгоритм буде мати той же недолік, що й k-середніх, а саме необхідність зазначення кількості кластерів. Для вирішення цієї проблеми було запропоновано наступний підхід. Проводиться побудова гістограми щільності розподілу для кожної з змінних відібраних для кластеризації. В результаті підрахунку кількості точок, навколо яких групуються значення можна отримати оцінку кількості кластерів у вибірці як найбільша для усіх змінних вибірки кількість локальних максимумів на гістограмі. Для вирішення питання оптимальної кількості стовпців гістограми можна використовувати формулу Скотта, формулу Фрідмана-Діаконіса або аналогічні їм. При реалізації алгоритму була використана формула Скотта [40], за рахунок більш низької

обчислювальної вартості в порівнянні з формулою Фрідмана-Діаконіса. Таким чином побудова і аналіз гістограм відбуватимуться за лінійний час.

Введення розрахунку значення функції приналежності не можна проводити тим же шляхом, що використовується в С-середніх, так як в С-середніх обчислення нового положення центроїда відбувається після накопичення інформації, а не по ходу. У разі, коли положення центроїда змінюється в міру приєднання точок, розраховане значення функції належності втратить свою актуальність у зв'язку зі зміною положення центроїда. В такому випадку значення функції приналежності слід розраховувати вже після формування кластерів. Крім того, функцію приналежності слід модифікувати таким чином, щоб розрахунок проводився без урахування становища центроїда, так як даний підхід передбачає формування кластерів гіперсферичної форми, в той час як реально кластери матимуть стрічкоподібну форму. Можливо кілька варіантів:

1) Використання середньої відстані від досліджуваної точки до всіх інших точок кластера:

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{\frac{1}{N_j - 1} \sum_{l=1}^{N_j} \left\| x_i - x_l^{(j)} \right\|}{\frac{1}{N_k} \sum_{l=1}^{N_k} \left\| x_i - x_l^{(k)} \right\|} \right)^{\frac{1}{m-1}}}.$$

2) Використання відстані від досліджуваної точки до «сліду» сформованого з попередніх положень центроїда:

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{\left\| x_i - t_{c_j} \right\|}{\left\| x_i - t_{c_k} \right\|} \right)^{\frac{1}{m-1}}}.$$

3.2. Перевірка роботи алгоритму на тестовій вибірці

Перевірка роботи алгоритму проводилася на наборах даних «Іриси Фішера» [41]. Набір даних «Іриси Фішера» містить довжину та ширину зовнішньої та внутрішньої долі оцвітини для кожного із 150 ірисів трьох видів. У вибірці присутньо по 50 ірисів кожного виду (рис 3.1). Цей набір даним є класичним для тестування різноманітних класифікаційних та кластеризаційних алгоритмів з 1936 року, коли Рональд Фішер продемонстрував на його прикладі розроблений ним метод дискримінантного аналізу.

В результаті проведення оцінки кількості кластерів за допомогою розробленої процедури було встановлено, що в наборі даних присутні 3 кластера. При кластеризації набору даних «Іриси Фішера» був отриманий наступний результат (табл. 3.1).

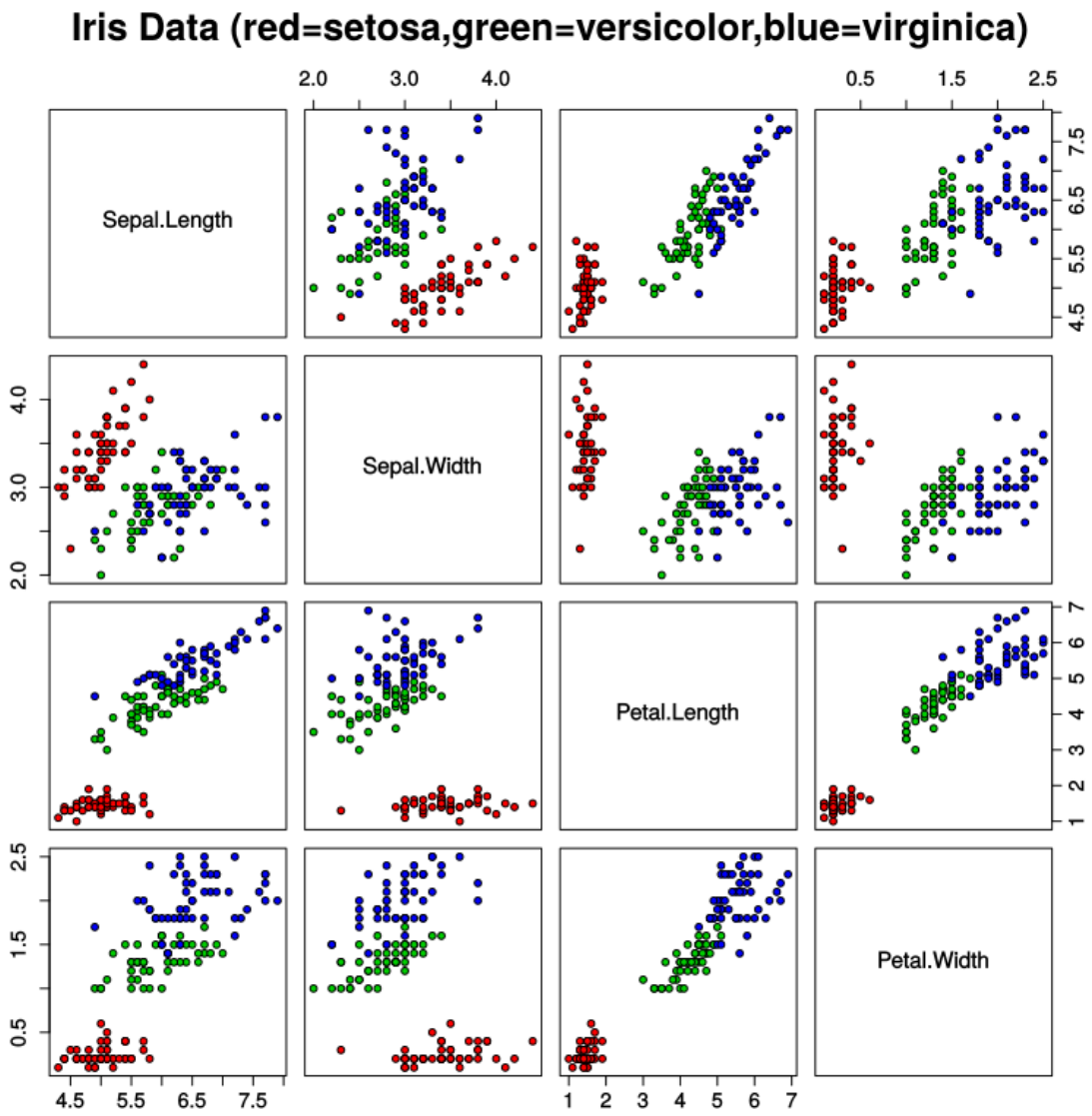


Рисунок 3.1 Діаграма розсіювання “Ірисів Фішера”.

Таблиця 3.1

Результат кластеризації

		Реальні			Сума
		0	1	2	
Результат	0	36	0	0	36
	1	14	50	0	64
	2	0	0	50	50
Сума		50	50	50	150
Доля розпізн., %		72	100	100	

Стовбчаста діаграма результату кластеризації зображена на рис. 3.2.

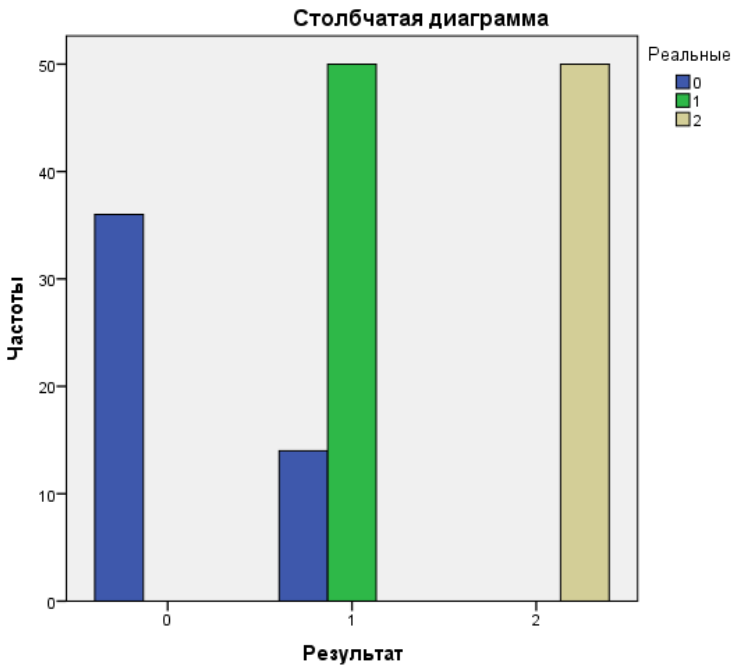


Рисунок 3.2 Діаграма результату кластеризації.

Як можна бачити, результат кластеризації виявився близький до реально існуючих груп. Значення F1-міри було отримано з використанням macro-averaging [42] і склало 0.92.

Для порівняння було використано метод Варда, оскільки він, подібно до розробленого методу, може будувати кластери несферичної форми.

Оскільки питання кількості кластерів у ієрархічному алгоритмі є досить складною темою, що не розглядалася у ході дослідження, кількість кластерів буде задано відповідно до кількості наявних груп.

У результаті проведення кластеризації методом Варда був отриманий наступний результат (табл 3.2).

Таблиця 3.2

Результат кластеризації методом Варда

		Реальні			Сума
		0	1	2	
Результат	0	33	0	0	33
	1	0	50	0	50
	2	17	0	50	67
Сума		50	50	50	150
Доля розпізн., %		66	100	100	

Стовбчаста діаграма результату кластеризації зображена на рис. 3.3.

Як можна бачити, результат отриманий методом Варда був дуже близьким до результату отриманого розробленим алгоритмом, хоча, як можна бачити з дендрограми (рис 3.4), якщо б не було задано реальну кількість груп, то метод Варда об’єднав би усі об’єкти в два кластера. Значення F_1-міри для метода Варда 0.90. Але не слід забувати, що ієрархічні алгоритми потребують велику кількість пам’яті для зберігання матриці відстаней або матриці подібності.

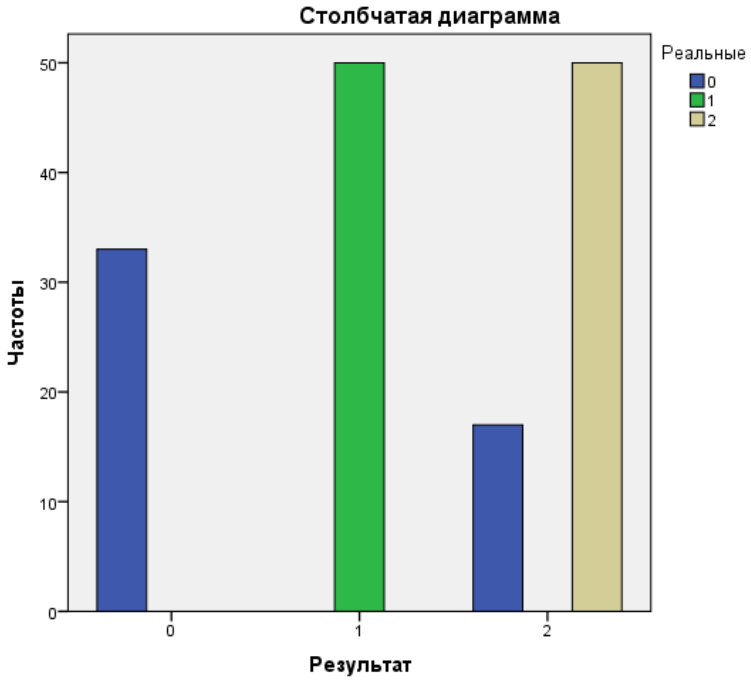


Рисунок 3.3. Стовбчаста діаграма результату кластеризації методом Варда.

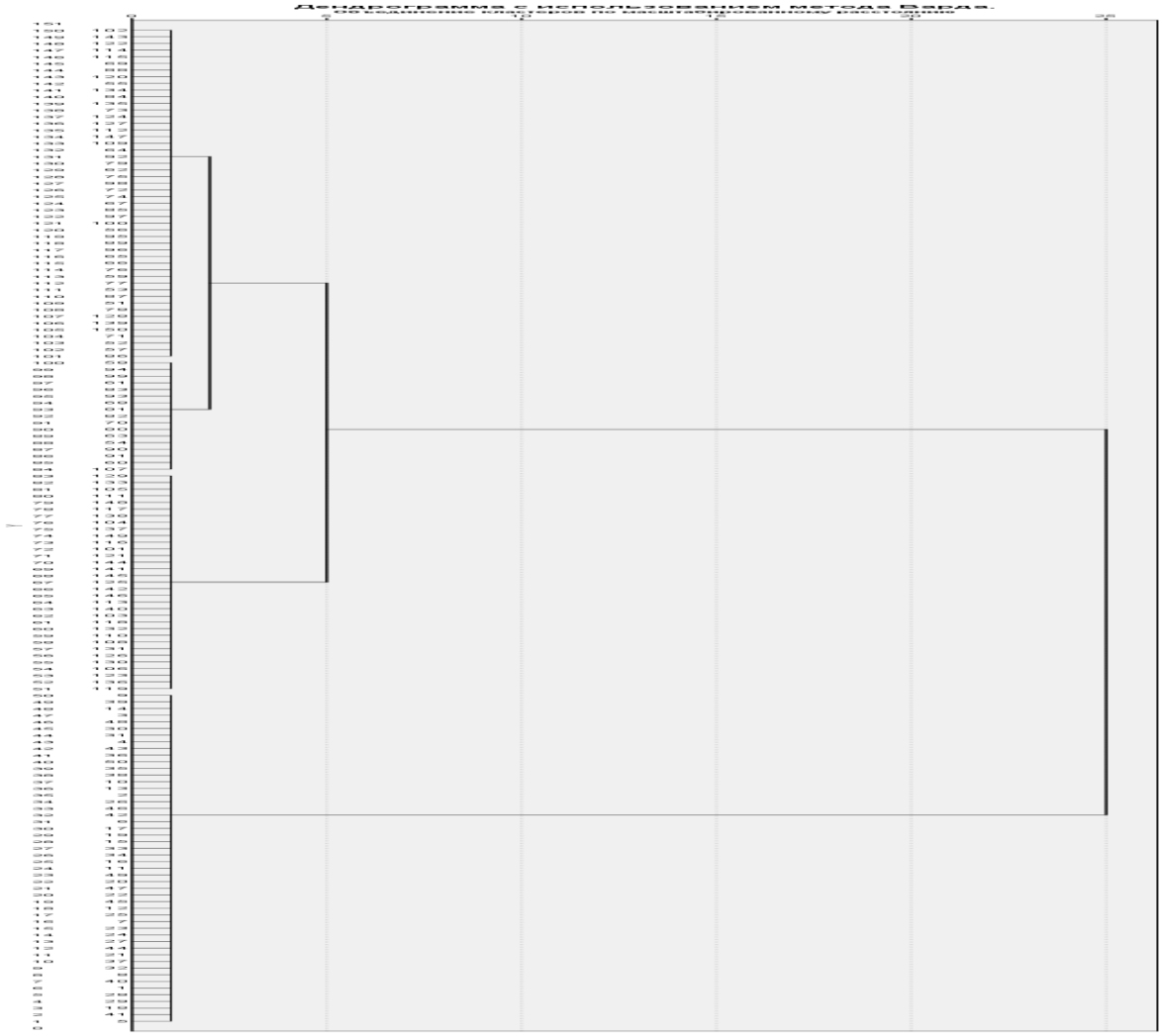


Рисунок 3.4. Дендрограма результату кластеризації методом Варда.

Кожна ітерація ієрархічного алгоритму породжує нову матрицю тому алгоритм потребує додаткової пам’яті. Розроблений алгоритм потребує додаткової пам’яті лише у випадку зберігання попередніх позицій центроїда кластера.

Висновки до розділу 3

У даному розділі описано розроблений алгоритм нечітких k-середніх з обмеженою масою робочої області. Проведено перевірку роботи алгоритма на тестовій вибірці. Для порівняння було обрано метод кластеризації Варда, що вирішує задачу подібно до тієї, що вирішує розроблений алгоритм. Результат розробленого алгоритму був

подібний до отриманого алгоритмом Варда, з відмінностями в сторону більшої якості розбиття. Розроблений алгоритм використовує меньше пам’яті та не потребує перерахунку матриці відстаней на кожному кроці роботи алгоритму.

РОЗДІЛ 4
ПРОГРАМНА РЕАЛІЗАЦІЯ МОДИФІКОВАНОГО АЛГОРИТМУ K-СЕРЕДНІХ

4.1. Формування вимог до програмного продукту

Метою розробки є виконання програмної реалізації нечіткого алгоритму k-середніх з обмеженою масою робочої області.

- Робота в поширених операційних системах.
- Можливість збереження результату в базу даних.
- Візуальне відображення побудованого розбиття.
- Оцінка якості отриманого розбиття.
- Можливість регулювання параметрів алгоритму кластеризації.

4.2. Проектування програмного продукту

Модель життєвого циклу.

Модель життєвого циклу є структурою, що складається з процесів, робіт та задач, які включають в себе розробку, експлуатацію та супровід програмного продукту та охоплюють життя системи від визначення вимог до припинення її експлуатації [43]. Вибір життєвого циклу є важливим кроком при створенні нової системи і його вибір на подальшу долю проекта складно переоцінити.

Однією з найбільш очевидних є каскадна модель, також відома як модель водоспаду. Її особливістю є те, що перехід на наступний етап здійснюється лише після завершення усіх робіт на попередній стадії. Кожна стадія закінчується одержанням результатів, що є вхідними даними для наступної стадії, та випуском повного комплекту документації. Вимоги до ПЗ, визначені на стадії формування вимог, документуються у вигляді

технічного завдання і фіксуються на весь час процесу розробки. Критерієм якості розробки для такої моделі є точність виконання вимог технічного завдання. Повернення на попередні стадії у даній моделі не передбачається.

Переваги застосування каскадної моделі:

- на кожній стадії формується закінчений набір проектної документації, яка відповідає критеріям повноти й узгодженості;
- виконання робіт у логічній послідовності дає змогу планувати терміни завершення всіх робіт і відповідні витрати.

Ця модель добре зарекомендувала себе при побудові інформаційних систем, для яких на самому початку розроблення можна досить точно і повно сформулювати усі вимоги.

Недоліки цієї моделі викликані насамперед тим, що реальний процес створення ПЗ ніколи цілком не укладався в жорстку схему. Процес створення ПЗ часто має ітераційний характер: результати чергової стадії викликають зміни у проектних рішеннях, що прийняті на попередніх стадіях.

Рисунок 4.1. Каскадна модель життєвого циклу.

Крім того, результат розробки можна побачити лише після повного завершення усіх робіт над проектом, тому неможливо змінити вимоги, які зазвичай не є повністю визначеними, що особливо актуально для великих проектів.

Інкрементна стратегія – це розробка ІС з лінійною послідовністю стадій, але с запланованим покращенням продукта з кожною новою версією (рис. 4.2).

Рисунок 4.2. Інкрементна модель життєвого циклу.

На початку роботи визначаються основні вимоги до ІС, після чого виконується її розробка у вигляді послідовності версій. При цьому кожна версія є закінченим продуктом. Кожна із наступних версій розширює можливості системи доки не буде отримана повна система.

Подібну модель використовують коли у замовника не вистачає фінансів щоб профінансувати великий проект відразу або коли розробник не має ресурсів для реалізації проекту за короткий строк.

Така модель життєвого циклу добре показує себе за умови чіткої уяви того, що має собою являти готовий продукт, як зі сторони замовника, так і розробника.

Переваги даної моделі подібні до переваг каскадної, але результат роботи можна побачити раніше, що дозволяє змінити вимоги до розробки або взагалі відмовитися від неї.

Ідея спіральної стратегії (рис 4.3), яку також називають еволюційною полягає у розробці послідовності версій коли на початку проекта визначені не всі вимоги, що уточнюються у процесі розробки нових версій продукту.

Рисунок 4.3. Спіральна модель життєвого циклу.

Еволюційна модель характерна для розробки новаторських систем, де на початку проекта замовник та розробник не мають чіткої уяви про те, яким має бути фінальний продукт або впевненості у тому, що проект буде успішно реалізовано. У таких випадках приймається рішення про розробку системи по частинам з можливістю зміни вимог або відмови від її подальшого розвитку.

Спіральна модель має наступні переваги:

- Дозволяє швидше продемонструвати користувачам системи готовий продукт для уточнення та доповнення вимог.
- Модель допускає значні зміни вимог при розробці ІС.
- Забезпечує високу гнучкість в управлінні проектом.
- Дозволяє виправляти помилки та слабкі місця системи у ході розробки з кожною ітерацією, отримуючи таким чином більш стабільну та надійну систему.
- Проведення аналізу на кожній ітерації дозволяє проводити оцінку того, що має бути змінено у проекті, щоб покращити його якість на наступній ітерації.
- Проект може бути завершено з мінімальними фінансовими витратами.

У той самий час є кілька значних недоліків такого підходу:

- Він збільшує невизначеність стосовно перспектив розвитку проекта у розробника, що витікає із деяких переваг моделі.
- Більш складне планування використання часу та ресурсів для усього проекта в цілому. Для вирішення цієї проблеми необхідно обмежувати строки виконання кожного з етапів проекта і здійснювати перехід навіть коли виконано не всю заплановану роботу.

Для розробки програмного продукту було використано Каскадну модель, оскільки усі вимоги до програмного продукту були відомі зазделегідь.

Побудова ієрархічної структури та розрахунок нев'язки.

Побудовану ієрархічну структуру зображено на рис. 4.5. Розрахуємо нев'язку за допомогою MATLAB (рис.4.4)

```
>> n = 6; %количество вершин полученного графа
E = 6; %количество ребер полученного графа
Et = n-1; %количество ребер дерева
Ec = n*(n-1)/2; %количество ребер полного графа
Nev = (E-Et)/(Ec-Et); %расчет невязки
Nev %вывод значений невязки

Nev =

    0.1
```

Рисунок 4.4. Розрахунок нев’язки.

Рисунок 4.5. Ієрархічна структура ПП.

Як можна побачити, ми отримали дуже малу нев’язку, що прагне до нуля. Це свідчить про добре сплановану систему, яку доцільно розробляти.

Методологія IDEF3.

Для опису логіки взаємодії інформаційних потоків підходить IDEF3, також відома як workflow diagramming, - методологія моделювання, що

використовує графічний опис потоків інформації, взаємин між процесами обробки інформації та об'єктів, які є частиною цих процесів.

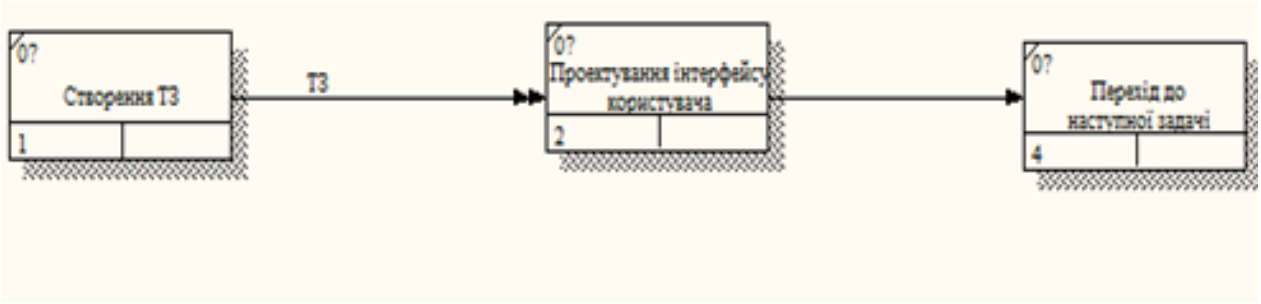


Рисунок 4.6. Діаграма IDEF3.

Методологія DFD.

Найважливішим способом опису процесу є діаграми потоків даних DFD (Data Flow Diagram). На рис. 4.7. зображено декомпозицію пункту «Аналіз вимог». Зовнішнім посиланням тут виступає замовник, який надає свої потреби. Сховищем даних є репозиторій стандартів.

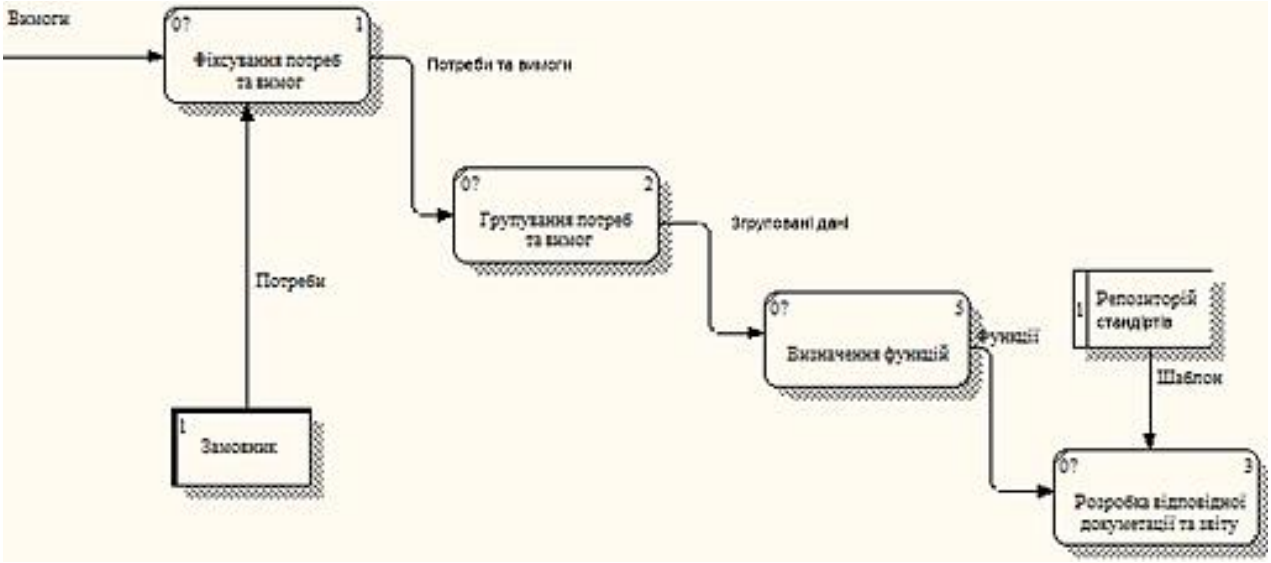


Рисунок 4.7. Діаграма DFD (аналіз вимог).

На рис. 4.8 зображено декомпозицію пункту «Аналіз аналогічних систем».

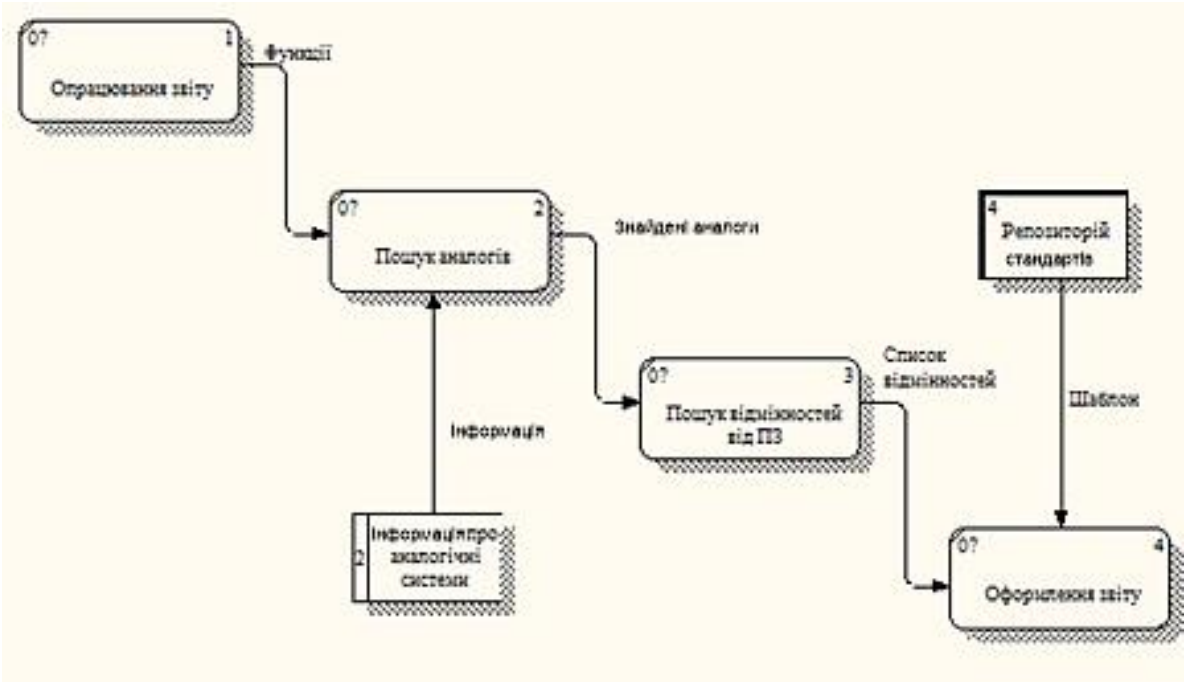


Рисунок 4.8. Діаграма DFD (аналіз аналогічних систем).

На рис. 4.9 зображено діаграму декомпозиції «Проектування», яка має наступні основні блоки:

- Проектування інтерфейсу.
- Проектування архітектури.

Для проектування інтерфейсу необхідно мати затверджений замовником дизайн.

Під час проектування архітектури необхідно знати вимоги замовника та можливі сценарії роботи з програмною системою.

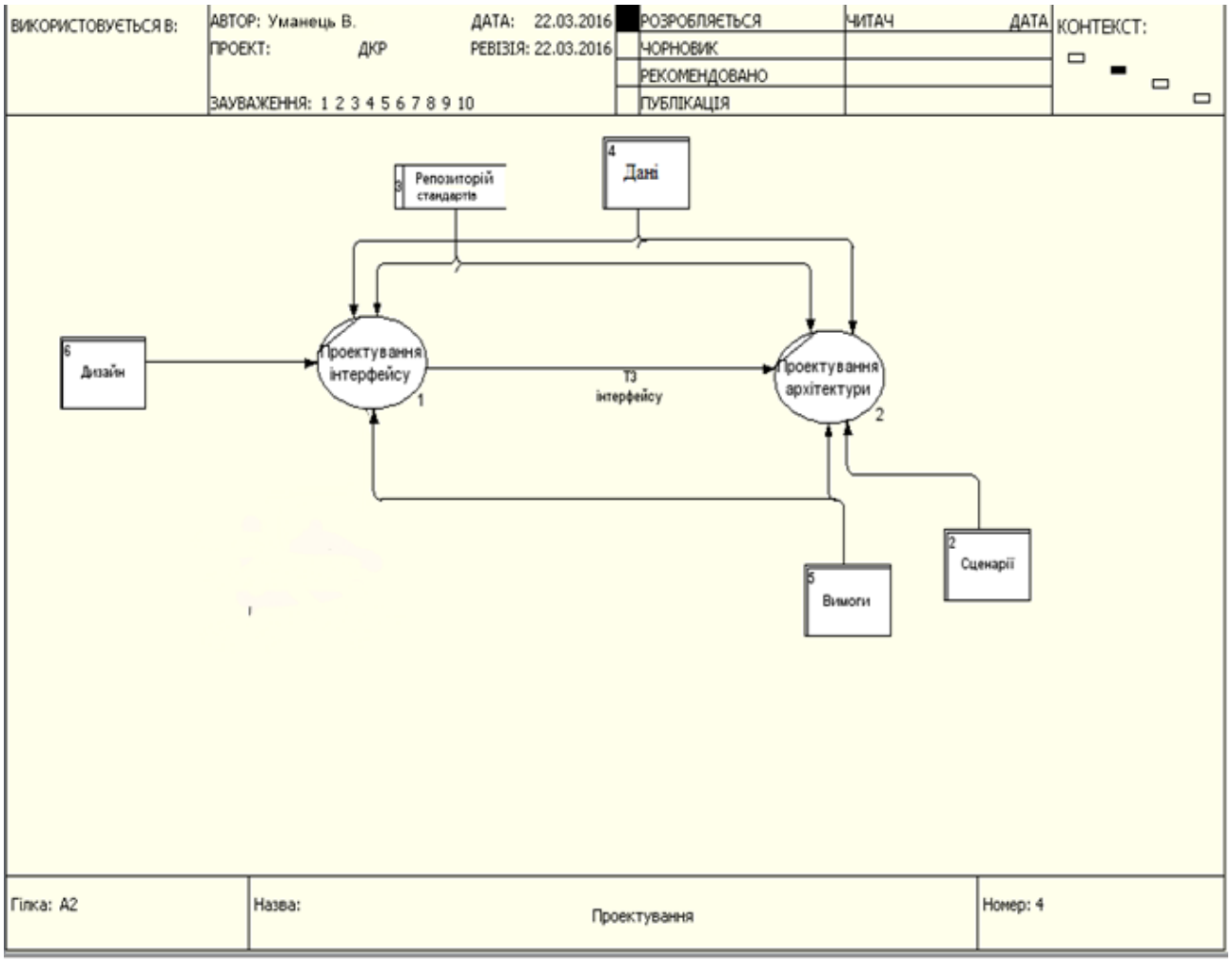


Рисунок 4.9. Діаграма декомпозиції «Проектування» (DFD).

Реалізація (рис. 4.10) складається з наступних блоків:

- реалізація інтерфейсу, модель котрого була побудована на етапі проектування;
- реалізація алгоритму кластеризації;
- реалізація виведення отриманих кластерів на графік.

Одне з найважливіший місць у розробці належить тестуванню. Оскільки необхідно створити якісну інструкцію з використання, протестувати інтерфейс та власне роботу системи.

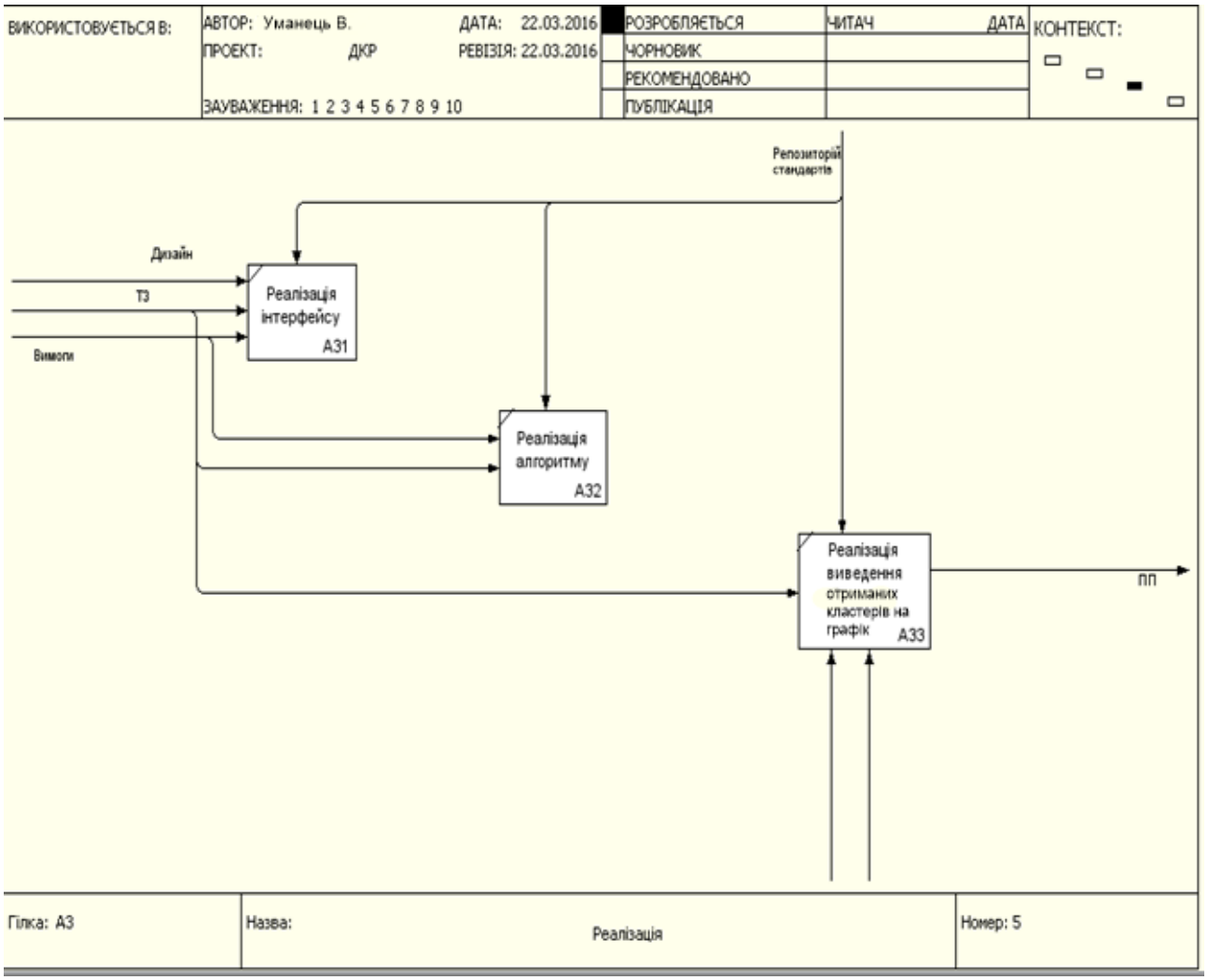


Рисунок 4.10. Діаграма декомпозиції «Реалізація».

Діаграми реалізації.

Діаграма компонентів — в UML, діаграма, на якій відображаються компоненти, залежності та зв'язки між ними.

Діаграма компонентів відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись.

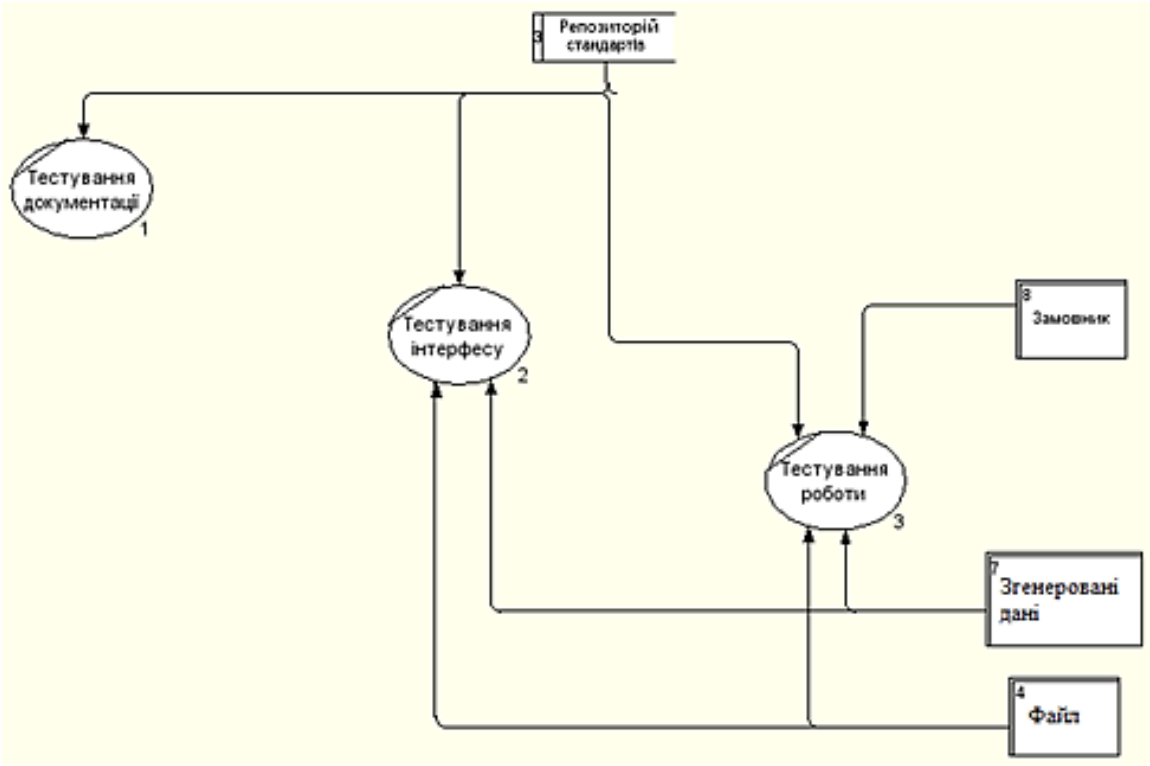


Рисунок 4.11. Діаграма декомпозиції «Тестування» (DFD).

Модуль програмного забезпечення може бути представлено як компоненту. Деякі компоненти існують під час компіляції, деякі — під час компонування, а деякі під час роботи програми.

Нижче наведені діаграми компонентів програмного комплексу (рис 4.12).

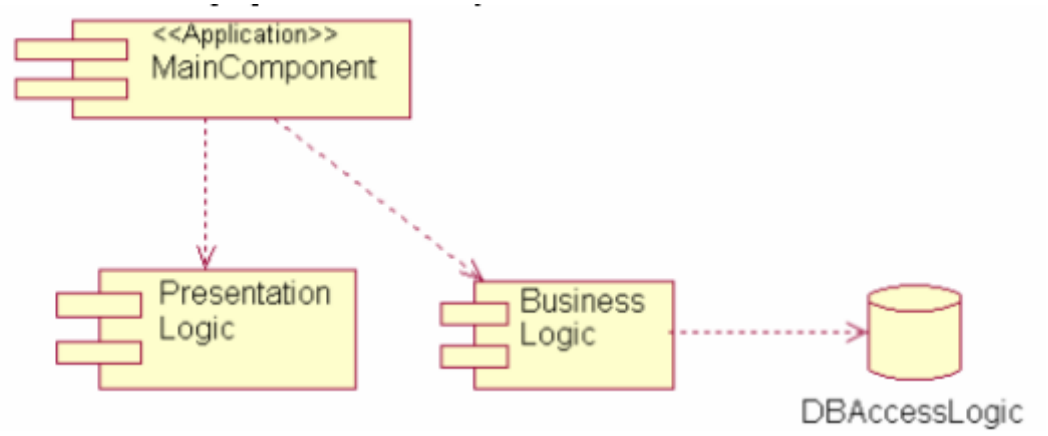


Рисунок 4.12. Діаграма компонентів програмного комплексу.

Діаграма дерева вузлів.

Діаграма дерева вузлів показує ієрархічну залежність робіт, але не взаємозв'язки між роботами.

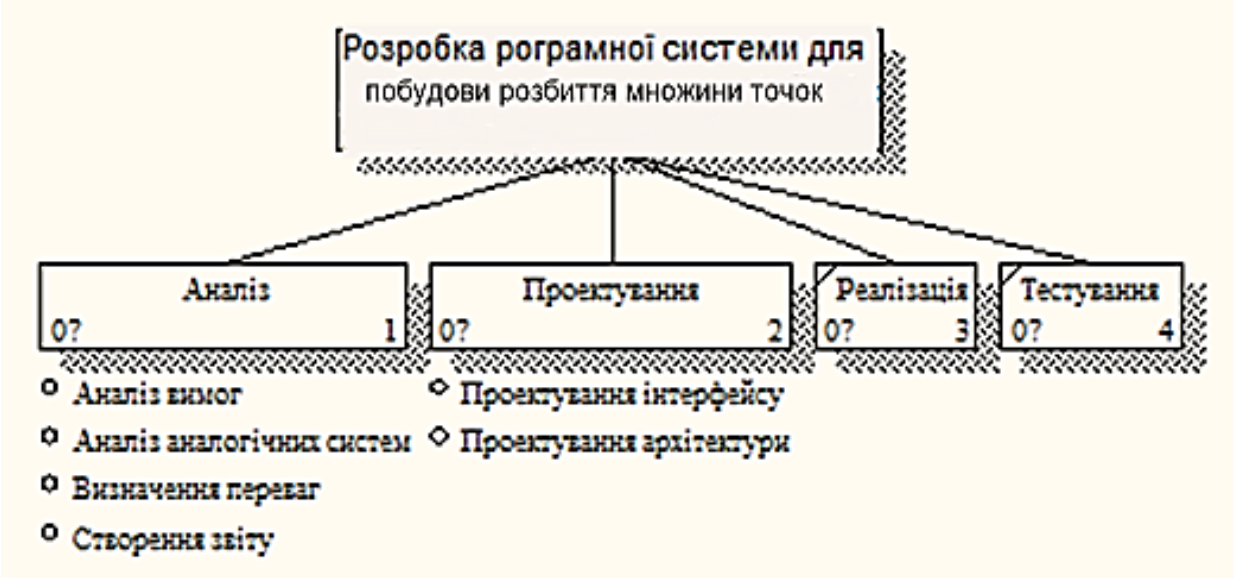


Рисунок 4.13. Діаграма дерева вузлів.

Контекстна діаграма.

Контекстна діаграма є найбільш верхньою діаграмою, на якій об'єкт моделювання представлений одним блоком з граничними стрілками [44].Стрілки на такій моделі відображають зв'язки об'єкта моделювання з навколишнім середовищем.

На контекстній діаграмі (рис 4.14) зображено процес побудови розбиття множини точок за допомогою нечіткого алгоритму k-середніх з обмеженою масою робочої області.

Вхідні дані включають в себе матрицю "об'єкт-властивості" та параметри, що необхідні для реалізації задачі. Вихідними даними є динамічний контейнер з утвореними кластерами.

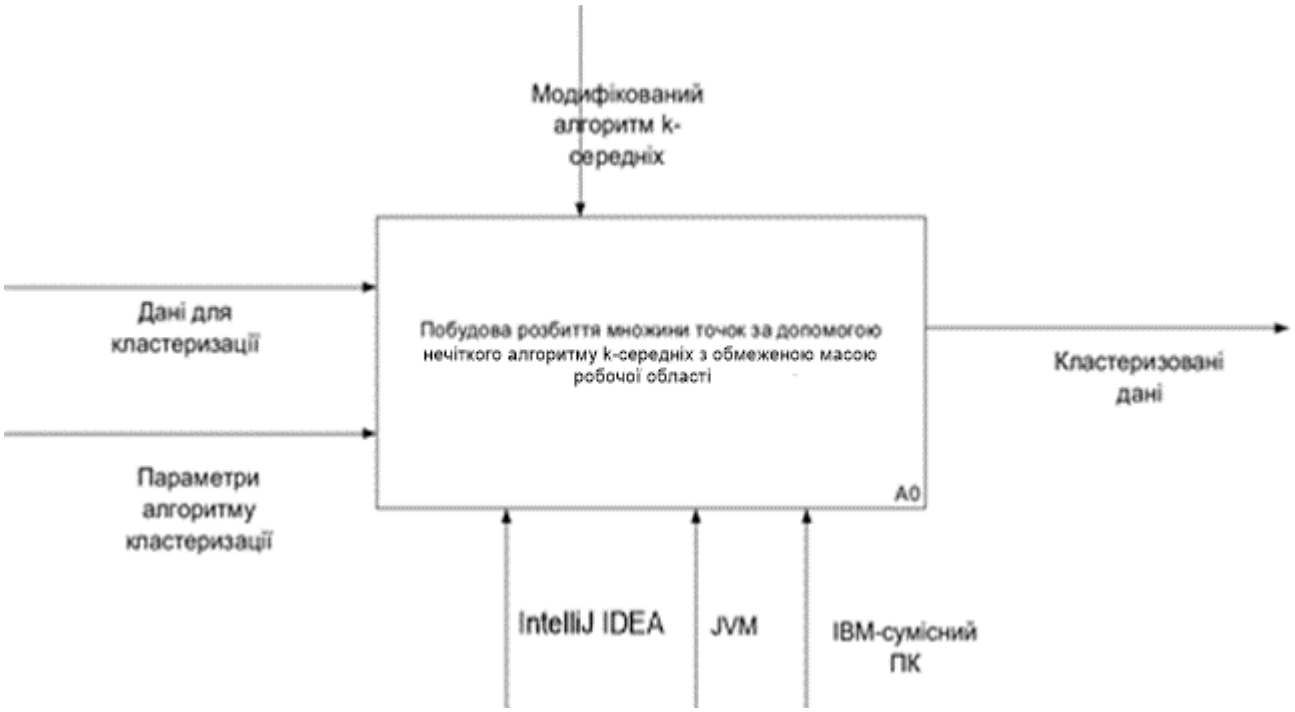


Рисунок 4.14. Контекстна діаграма.

Діаграма декомпозиції першого рівня (методологія IDEF0).

Нотація IDEF0 підтримує послідовну декомпозицію до необхідного рівня деталізації. Тому для наочного відображення підпроцесів, що включає в себе процес побудови розбиття заданої множини точок у вигляді просторових ліній, зручно використати діаграму декомпозиції (рис. 4.15). На діаграмі більш детально зображено процес побудови розбиття множини об’єктів за допомогою модифікованого алгоритму k-середніх для функціонально зв'язних кластерів. Процес було розбито на чотири підпроцеси, а саме: обробку даних, оцінку кількості кластерів, власне проведення кластеризації та збереження результату.

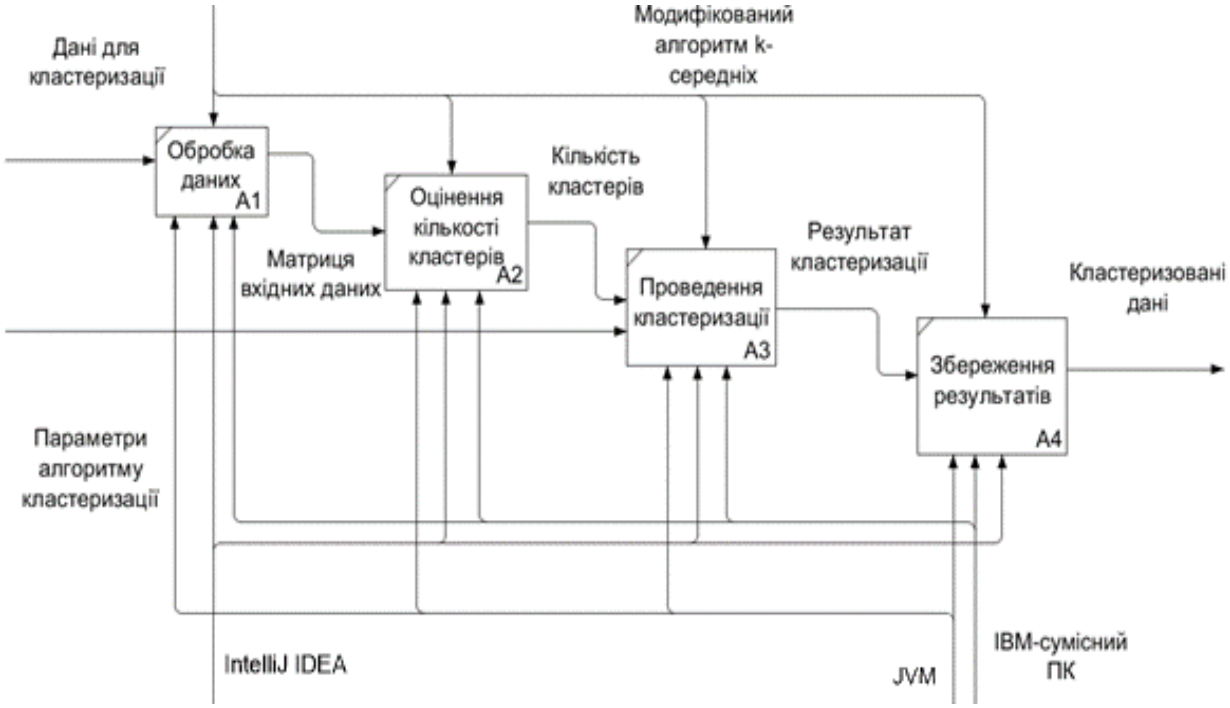


Рисунок 4.15. Діаграма декомпозиції першого рівня.

Діаграма сутність-зв'язок (ERD).

ERD діаграма згідно до завдань, що були сформовані на етапі системного аналізу та аналізу вимог, показана на рис. 4.16

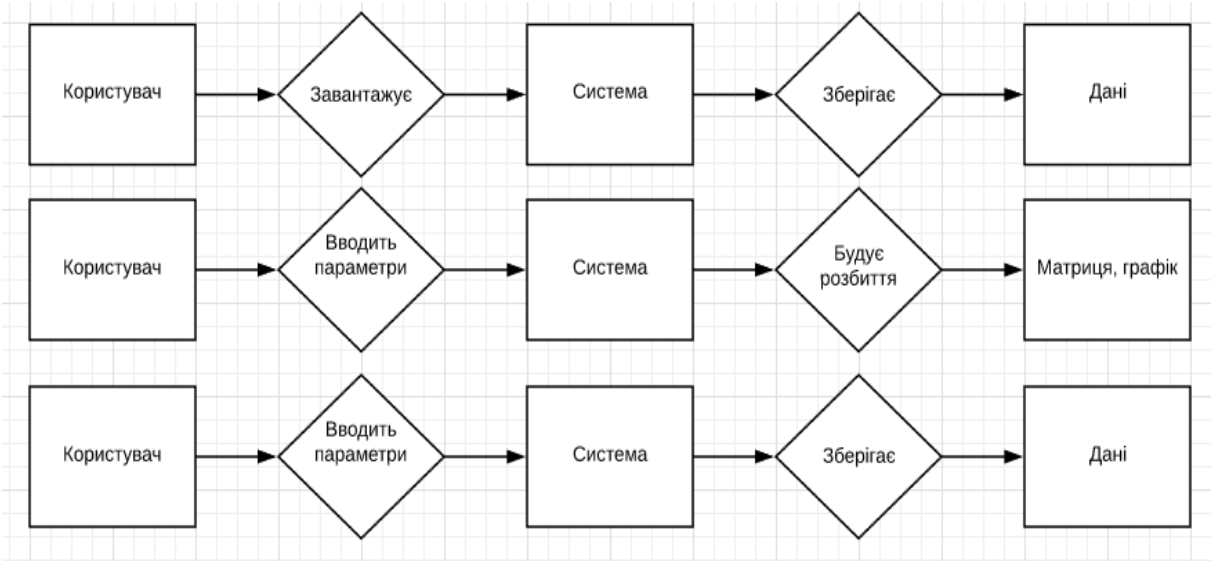


Рисунок 4.16. Діаграма сутність-зв'язок.

Діаграма потоку даних (DFD)

DFD діаграма згідно до завдань, що були сформовані на етапі системного аналізу та аналізу вимог, показана на рис. 4.17.

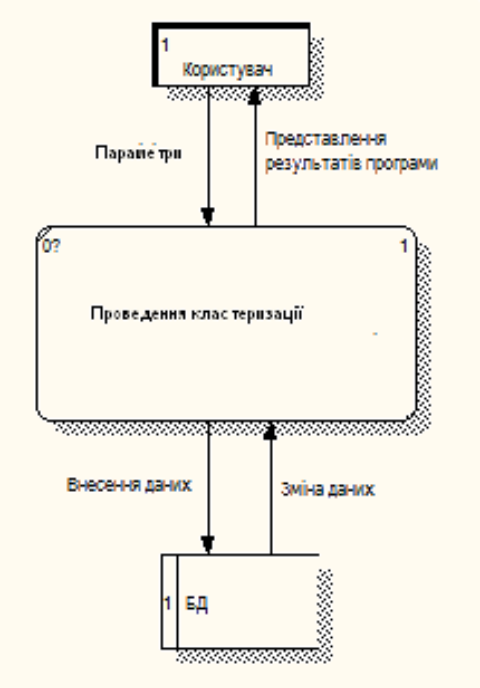


Рисунок 4.17. Діаграма потоку даних.

Діаграма функціонального моделювання (SADT).

SADT діаграма згідно до завдань, що були сформовані на етапі системного аналізу та аналізу вимог, показана на рис. 4.18.

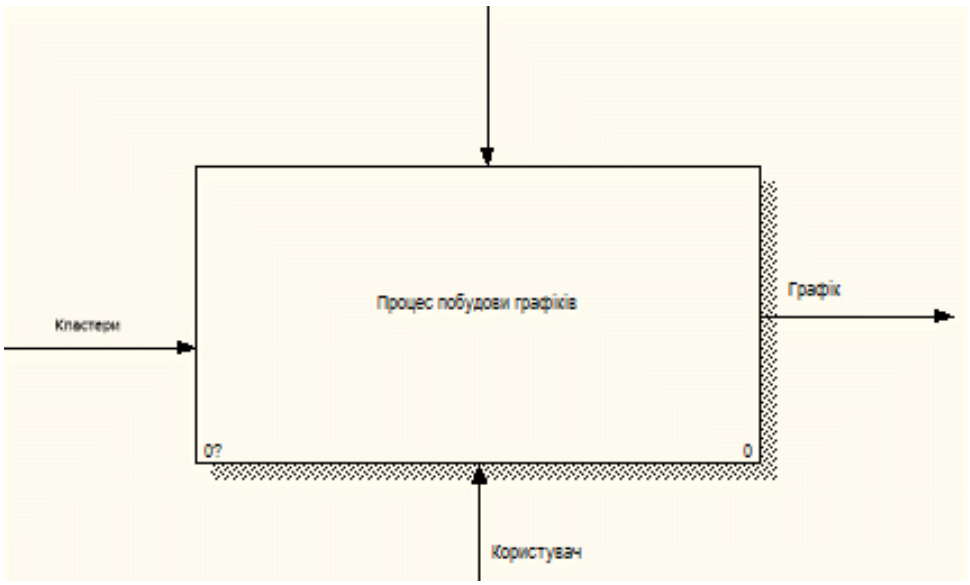


Рисунок 4.18. Діаграма функціонального моделювання.

Модель варіантів використання (Use Case)

Діаграма варіантів використання (Use Case) [45] відображає процеси, що відбуваються при виконанні алгоритму з точки зору акторів. Актором є будь-яка сутність, що взаємодіє з системою ззовні. Це може бути як людина, так і програма або технічний прилад. Варіант використання слугує для того, щоб описати сервіси, що система представляє актору. Окремі варіанти використання на діаграмі позначаються еліпсами, а актор позначається фігуркою людини. Одні варіанти використання можуть бути частиною інших варіантів, розширювати їх або бути більш узагальненою версією.

Оскільки учасником виконання алгоритму є тільки один актор – користувач, то всі процеси, що відбуваються, прив’язані тільки до нього (рис. 4.19). Актору надаються лише три варіанта роботи з системою, а саме: завантаження даних, побудова розбиття та збереження результатів.

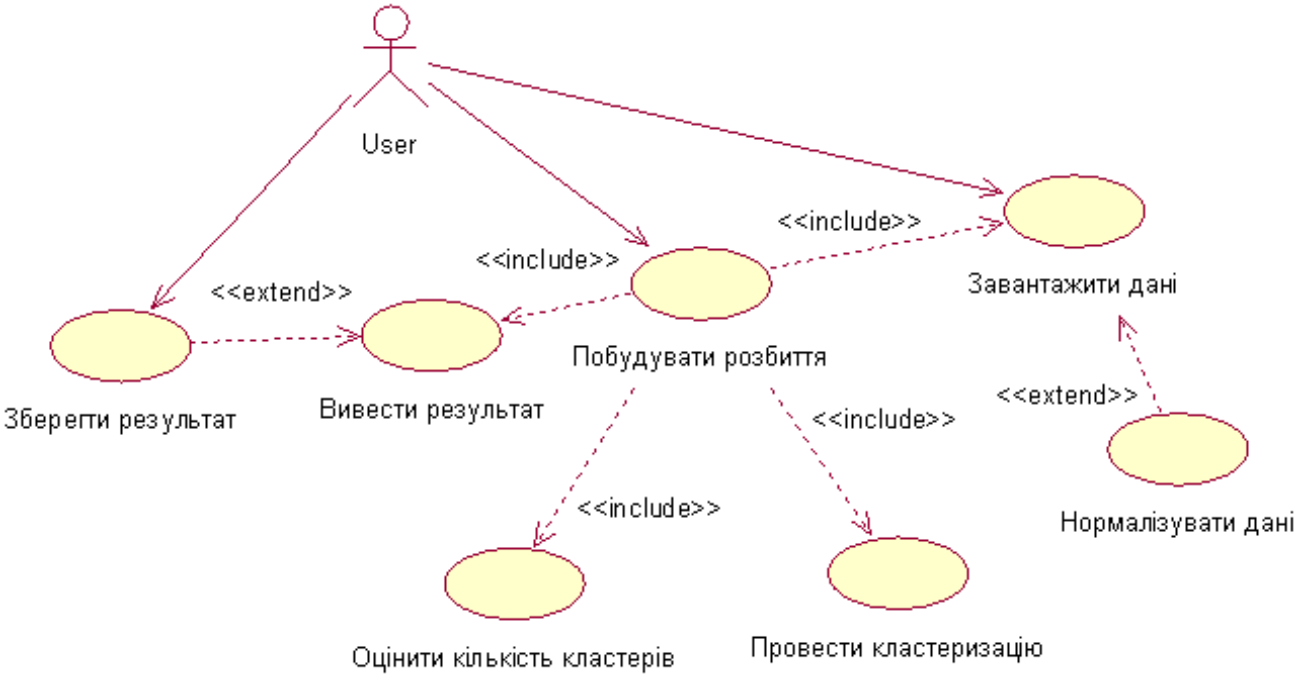


Рисунок 4.19. Діаграма Use Case.

Діаграма станів.

Діаграма станів [46] слугує для того, щоб пояснити роботу складних об'єктів та продемонструвати перехід об'єкта з одного стану в інший. Вона слугує для моделювання динамічних аспектів системи. Від інших діаграм діаграма станів відрізняється тим, що описує процес зміни стану лише для одного об'єкту.

На діаграмі станів (рис.4.20) відображено зв’язки і послідовність проходження всіх станів, від початку роботи системи і до кінця.

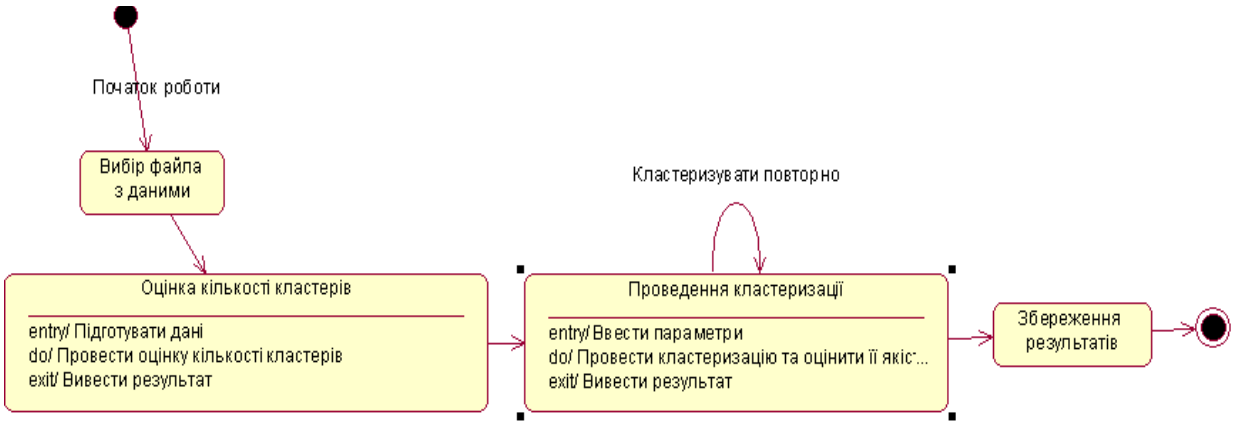


Рисунок 4.20. Діаграма станів.

Діаграма кооперації.

Діаграма кооперації [47] слугує для того, щоб описати логіку виконання складних операцій та продемонструвати набор об'єктів, що взаємодіють у реальному оточенні.

З діаграми кооперації (рис 4.21) видно, що користувач взаємодіючи з головним вікном програми, викликає запуск функцій окремих модулів. Користувач вибирає файл з даними, ініціює початок кластеризації та нові ітерації. Після виконання кластеризації система виводить на екран результати та зберігає результат при необхідності.

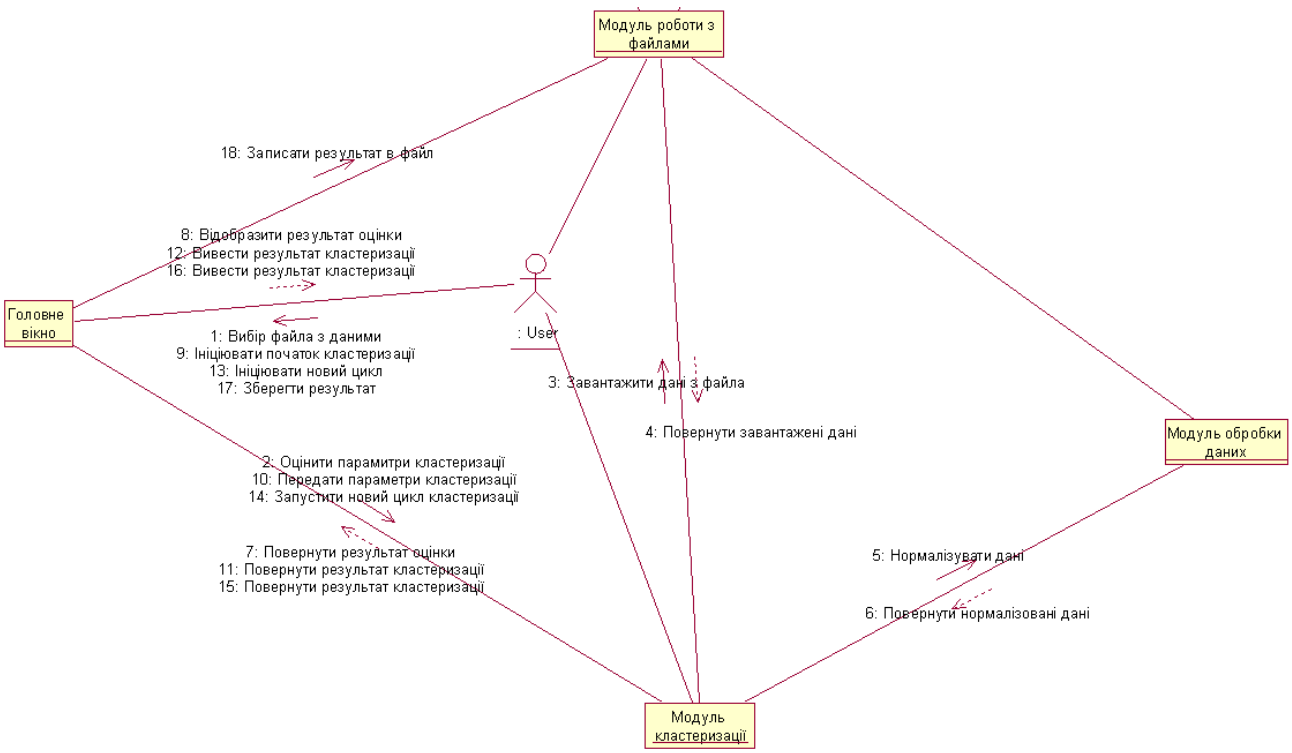


Рисунок 4.21. Діаграма кооперації.

Діаграма послідовності.

Діаграми послідовності [48] відноситься до діаграм взаємодії UML, що описують аспекти поведінки системи, але розглядають взаємодію об'єктів у часі. Вона може слугувати для більш детального опису логіки варіантів використання. На діаграмі послідовності об'єкти взаємодіють об'єднуючись між собою інформацією у вигляді повідомлень, що відповідає принципам об'єктно-орієнтованого програмування.

Діаграма послідовності на рис. 4.22 відображає взаємодію логічних елементів системи між собою в процесі виконання кластеризації даних. На діаграмі зображені всі варіанти використання актором системи, що були вказані раніше на діаграмі варіантів використання.

Обмін повідомленнями відбувається між актором та чотирьома модулями, а саме: головним вікном, модулем роботи з файлами, модулем обробки даних та модулем кластеризації. Задачею головного вікна є взаємодія з користувачем. В залежності від варіанту використання користувачем системи даний модуль передає відповідне повідомлення до

того модуля, що буде виконувати поставлену задачу. Модуль роботи з файлами відповідає за зчитування вхідних даних із файлу та збереження результатів роботи. Єдиною задачею модуля обробки даних є нормалізація даних поданих на вхід. Модуль кластеризації виконує кілька задач. По-перше, його основною задачею є побудова розбиття множини об’єктів. По-друге, задачею цього модуля є оцінка кількості кластерів. Результати роботи відображаються у головному вікні програми даючи можливість користувачеві керувати процесом кластеризації та зберегти результат роботи.

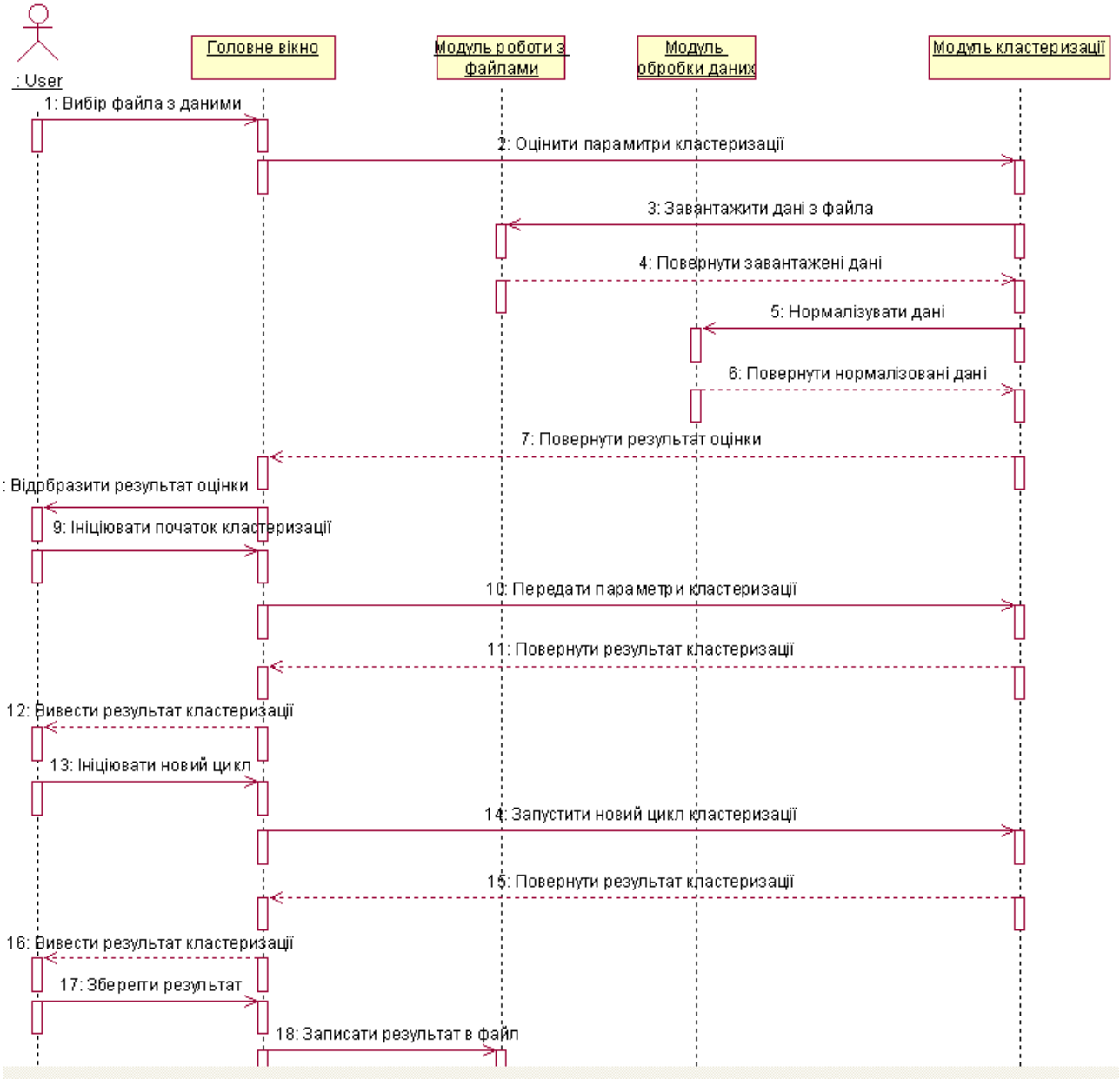


Рисунок 4.22. Діаграма послідовності.

Діаграма діяльності.

Діаграма діяльності [49] слугує для моделювання складного життєвого циклу об'єкта або сукупності об'єктів з переходами від одного виду діяльності до іншого. У тих випадках, коли діаграма описує діяльність багатьох об'єктів можливо використовувати "плавальні дорожки", що розподіляють різні види діяльності між тими об'єктами системи, що їх виконують. Крім того є можливість описання дій, що відбуваються паралельно та синхронізація одночасних операцій. Часто діаграми діяльності використовують для проектування процесів або операцій.

Зображена на рис. 4.23 діаграма діяльності відображає дії, що виконуються в процесі побудови розбиття множини точок. Програма у даному випадку розглядається не як складена структура, а як цілісний об’єкт. Тобто розглядається робота системи в цілому без урахування спеціалізації її модулів.

Рисунок 4.23. Діаграма діяльності.

4.3. Реалізація програмного продукту

Розроблений програмний продукт написано переважно на Java з використанням Scala у якості додаткової.

Метою розробки є виконання програмної реалізації нечіткого алгоритму k-середніх з обмеженою масою робочої області. Програмне забезпечення дозволить будувати розбиття множини точок на кластери несферичної форми.

Розроблений інтерфейс користувача є простим та інтуїтивно зрозумілим.

Вхідними даними при роботі з програмою можуть слугувати будь-які чисельні дані, задані як матриця "об'єкт-властивості.

4.4. Робота з розробленим програмними забезпеченням

Інтерфейс користувача розробленого програмного продукту представлено на рис. 4.24

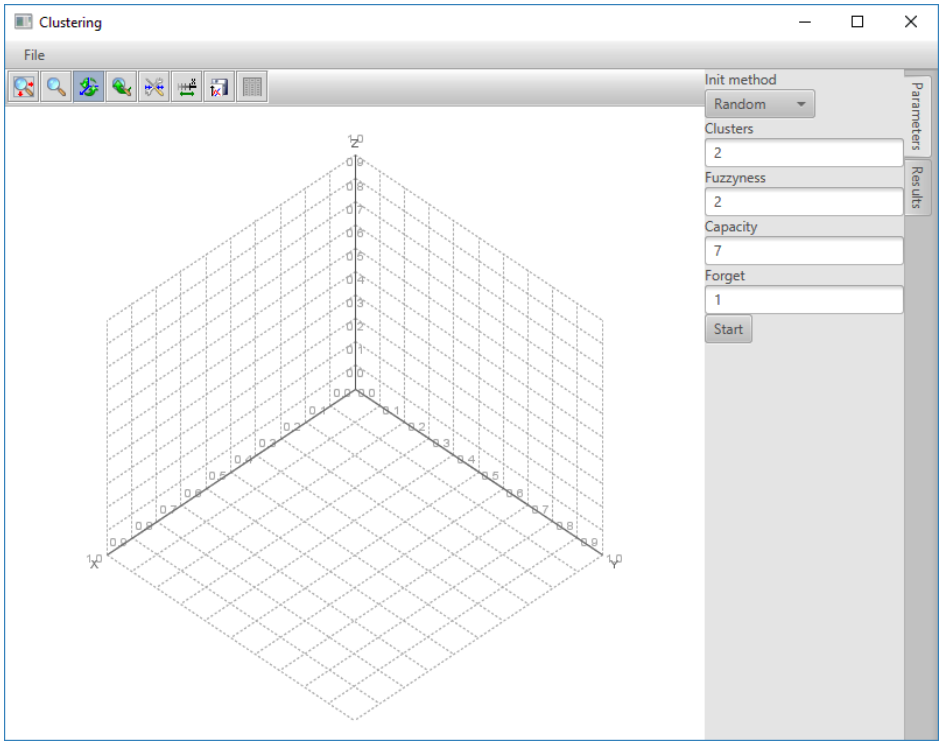


Рисунок 4.24. Інтерфейс програмного продукту.

Для того, щоб запустити розроблений програмний продукт необхідно використати спеціальний файл start.bat (рис. 25).

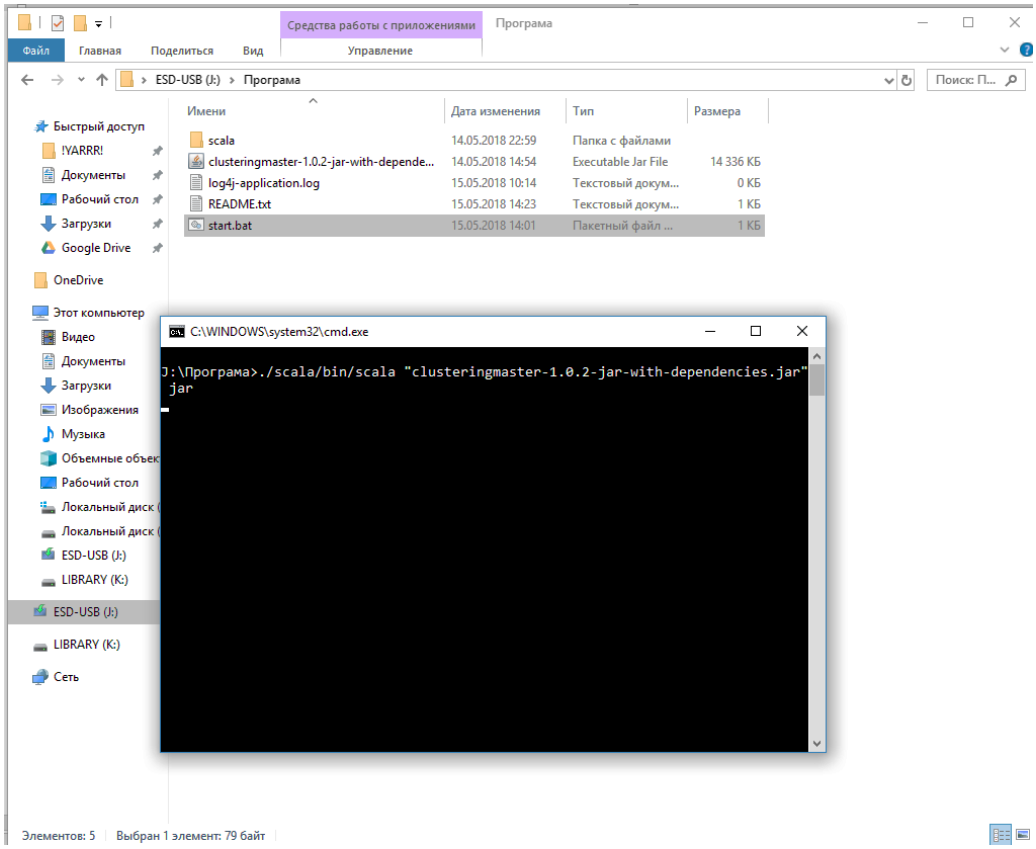


Рисунок 4.25. Інтерфейс програмного продукту.

Вікно можна умовно поділити на три частини: меню, параметри та графік. Перша частина "Меню" призначена для вибору джерела даних (рис 4.26). Також за допомогою "Меню" можна ініціювати завершення роботи програми.

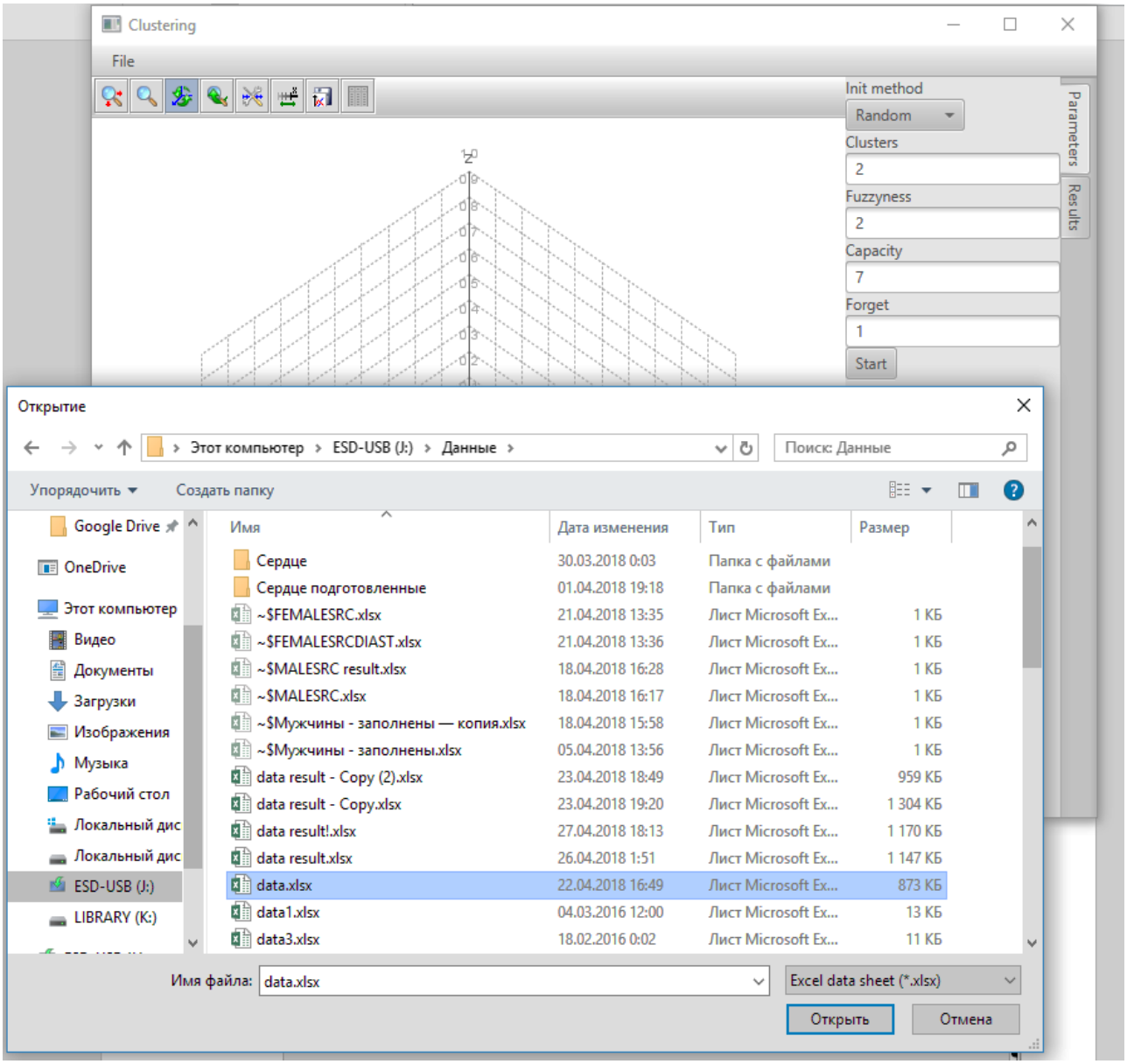


Рисунок 4.26. Приклад діалогового вікна для вибору вхідних даних.

Частина "Параметри" відповідає за налаштування параметрів та керування процесом кластеризації. Рекомендована кількість кластерів визначається автоматично після відкриття файлу з даними.

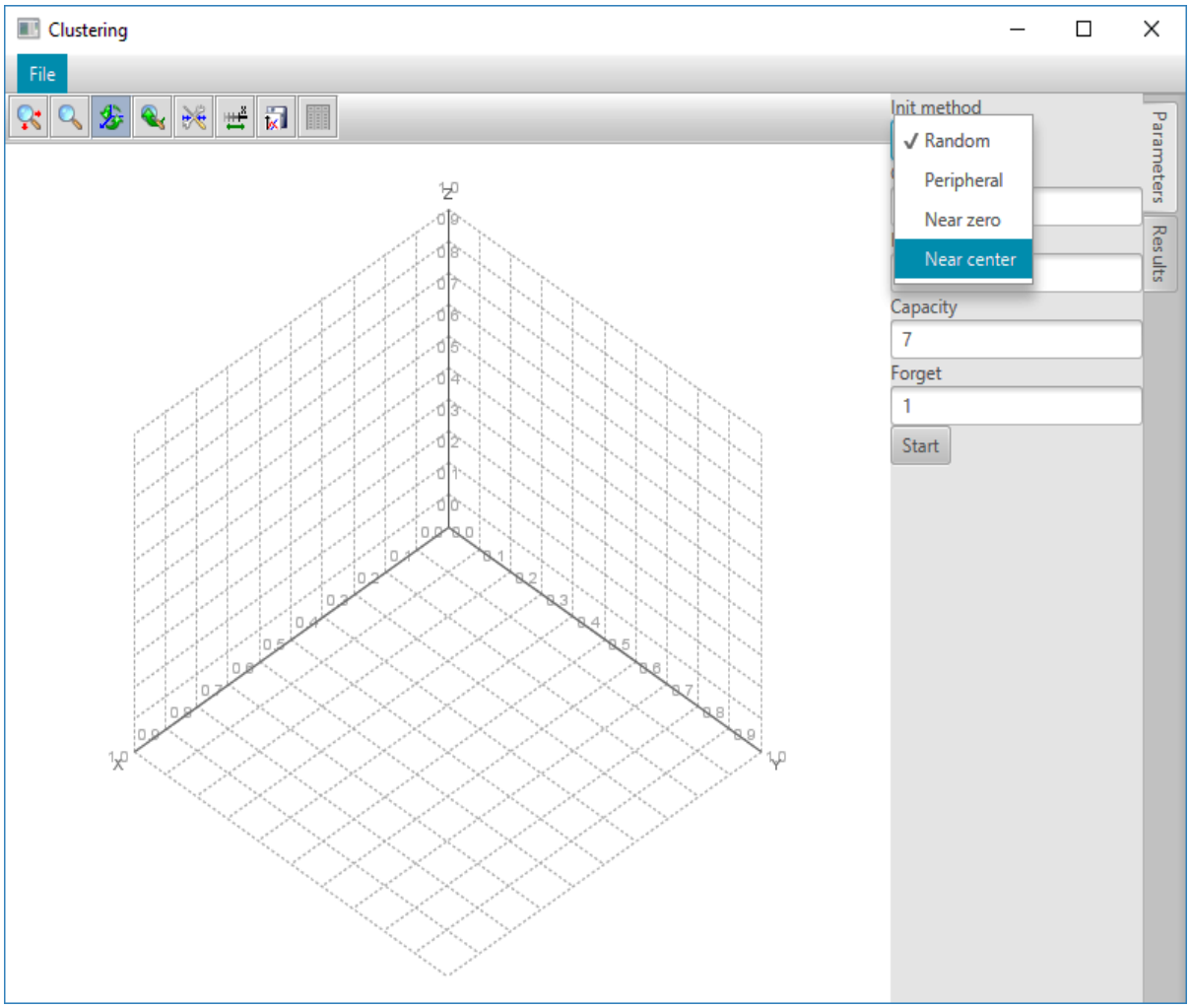


Рисунок 4.27. Вибір методу ініціалізації центроїдів.

Також тут здійснюється вибір методу ініціалізациі центроїдів (рис 4.27): випадково, на периферії, біля центру мас або біля початку координат. Для проведення кластеризації необхідно натиснути кнопку "Start".

Крім того для отриманого результату кластеризації відбувається розрахунок індекса силуета [35] окремо для кожного з кластерів для оцінки якості кластеризації. Дтя того, щоб переглянути значення індексів силуета, необхідно перейти на вкладку “Result” (рис 4.28).

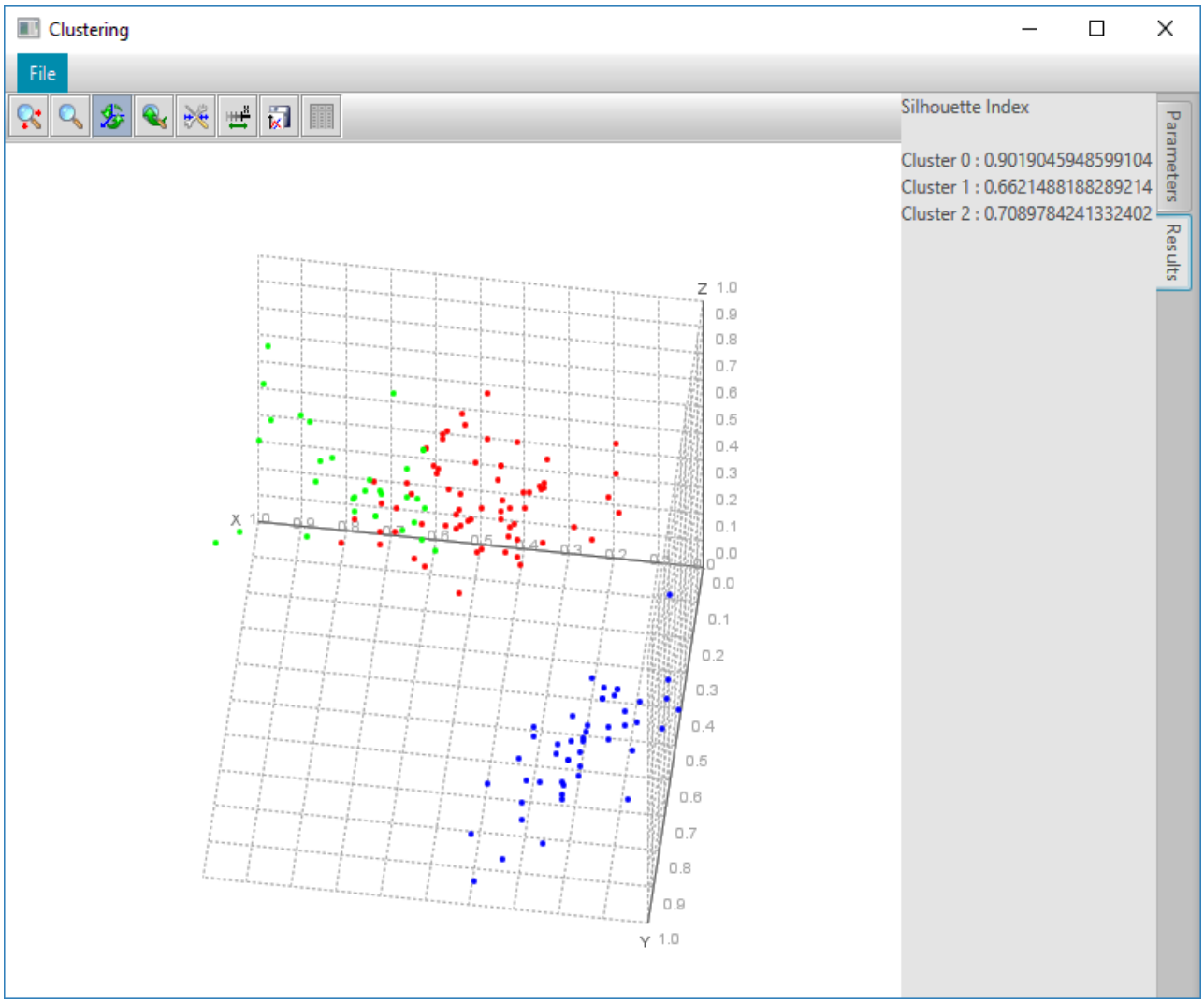


Рисунок 4.28. Індекс силуета.

Третя частина "Графік" слугує для візуального відображення результатів кластеризації. Для цього була використана бібліотека з відкритим вихідним кодом JMathPlot [50]. Кнопки над графіком слугують для виконання різноманітних маніпуляцій з отриманим результатом. Перша кнопка зліва, з зображенням лупи та стрілочок відповідає за переміщення графіка по горизонталі та по вертикалі. Друга кнопка відповідає за масштабування. Третя відповідає за поворот. Четверта відновлює початковий стан графіка. П’ята слугує для редагування шкали вимірювання для осей. Наступна кнопка перемикає ці шкали між фіксованою шкалою та автоматичним масштабуванням. Сьома кнопка слугує для збереження графіку до PNG (рис 4.29).

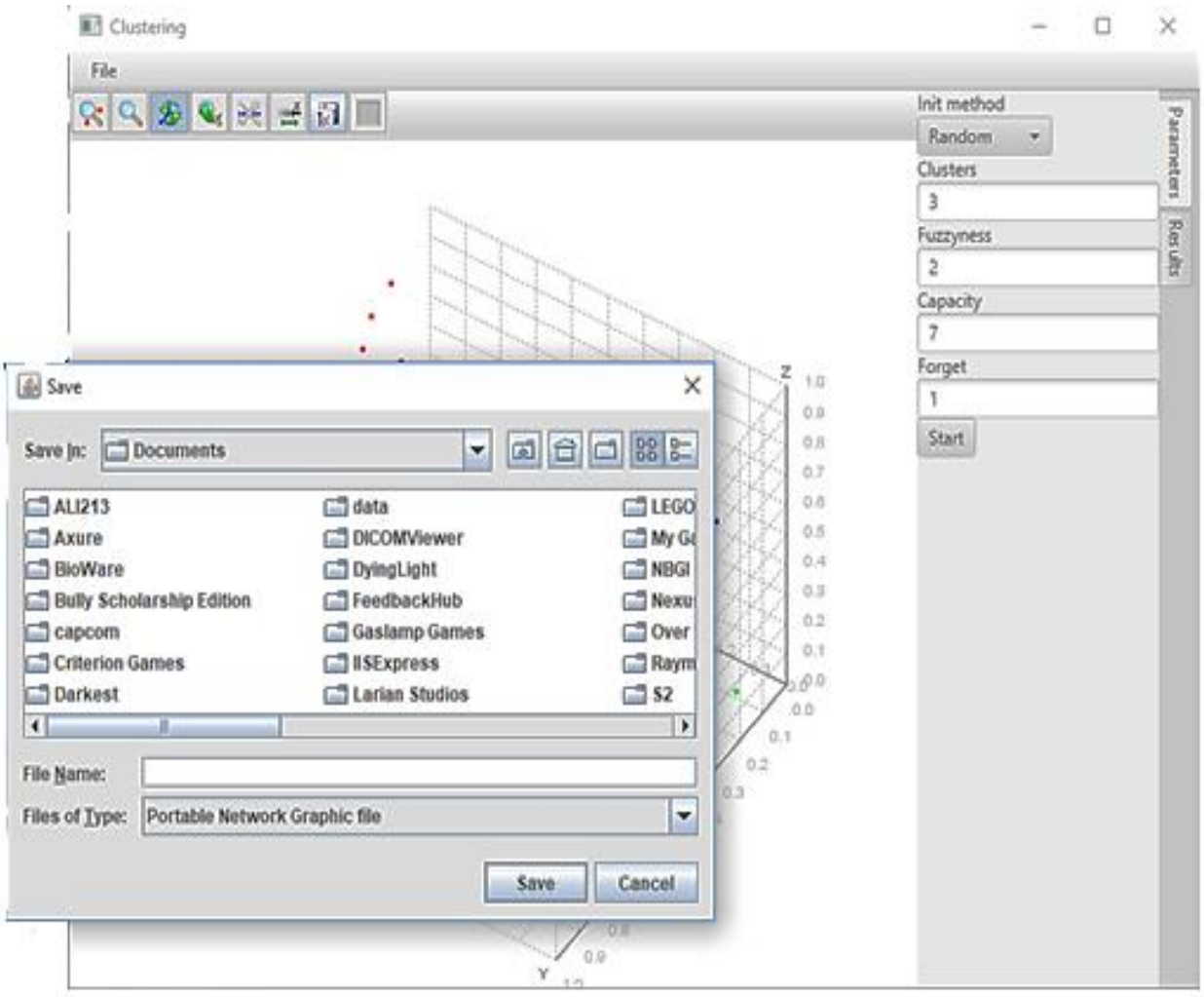
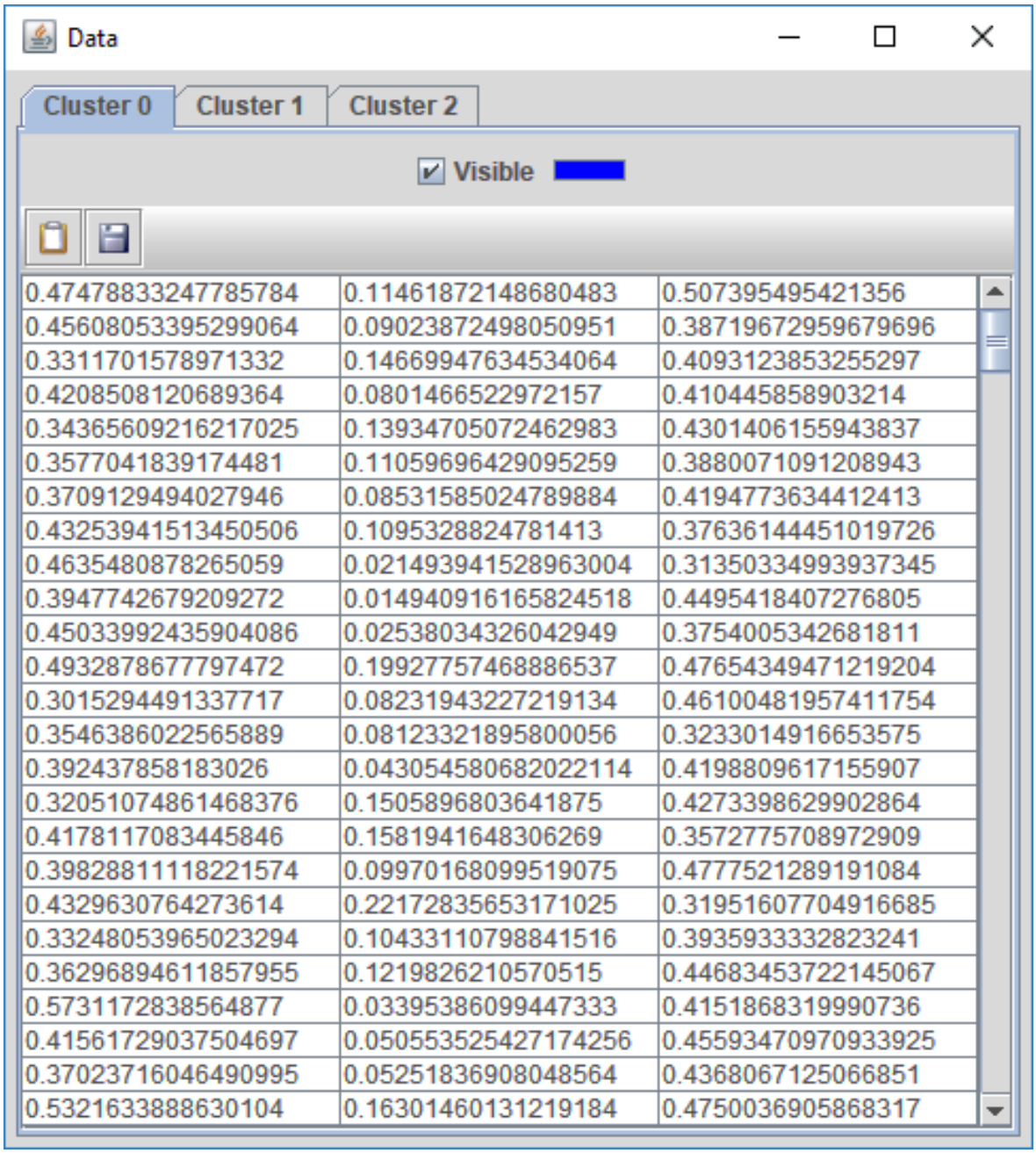


Рисунок 4.29. Збереження графіку до PNG файла.

І остання кнопка дозволяє переглянути точки, що увійшли до кожного з кластерів, а також змінювати колір, яким відображаються точки, що відносяться до різних кластерів та вмикати або вимикати відображення кластерів (рис 4.30). У даному вікні також можна зберегти дані окремого графіка в файл або скопіювати в буфер обміну.

Використана бібліотека надає широкі можливості для вибору кольору для відображення кластера. Є можливість задати колір у форматах: RGB, HSV, HSL, CMYK, або вибрати один із вже наданих кольорів на вкладинці Swatches (Рис 4.31)



0.47478833247785784	0.11461872148680483	0.507395495421356
0.45608053395299064	0.09023872498050951	0.38719672959679696
0.3311701578971332	0.14669947634534064	0.4093123853255297
0.4208508120689364	0.0801466522972157	0.410445858903214
0.34365609216217025	0.13934705072462983	0.4301406155943837
0.3577041839174481	0.11059696429095259	0.3880071091208943
0.3709129494027946	0.08531585024789884	0.4194773634412413
0.43253941513450506	0.1095328824781413	0.37636144451019726
0.4635480878265059	0.021493941528963004	0.31350334993937345
0.3947742679209272	0.014940916165824518	0.4495418407276805
0.45033992435904086	0.02538034326042949	0.3754005342681811
0.4932878677797472	0.19927757468886537	0.47654349471219204
0.3015294491337717	0.08231943227219134	0.46100481957411754
0.3546386022565889	0.08123321895800056	0.3233014916653575
0.392437858183026	0.043054580682022114	0.4198809617155907
0.32051074861468376	0.1505896803641875	0.4273398629902864
0.4178117083445846	0.1581941648306269	0.3572775708972909
0.39828811118221574	0.09970168099519075	0.4777521289191084
0.4329630764273614	0.22172835653171025	0.31951607704916685
0.33248053965023294	0.10433110798841516	0.3935933332823241
0.36296894611857955	0.1219826210570515	0.44683453722145067
0.5731172838564877	0.03395386099447333	0.4151868319990736
0.41561729037504697	0.050553525427174256	0.45593470970933925
0.37023716046490995	0.05251836908048564	0.4368067125066851
0.5321633888630104	0.16301460131219184	0.4750036905868317

Рисунок 4.30. Перегляд даних, що використовувалися для побудови графіку.

Вхідні дані для кластеризації мають бути спеціальним чином оформлені. Вхідні дані у таблиці Excel мають знаходитися на першій сторінці робочої книги та не мати підписів у першому рядку. Приклад вхідного файлу зображено на рис. 4.32. Дані не обов’язково повинні бути нормалізовані, оскільки нормалізація виконується безпосередньо під час проведення кластеризації. У файлі результату міститься копія вихідних

змінних, значення функції належності до кожного з кластерів та номер кластера, де значення функції належності є максимальним для даної точки (рис. 4.33).

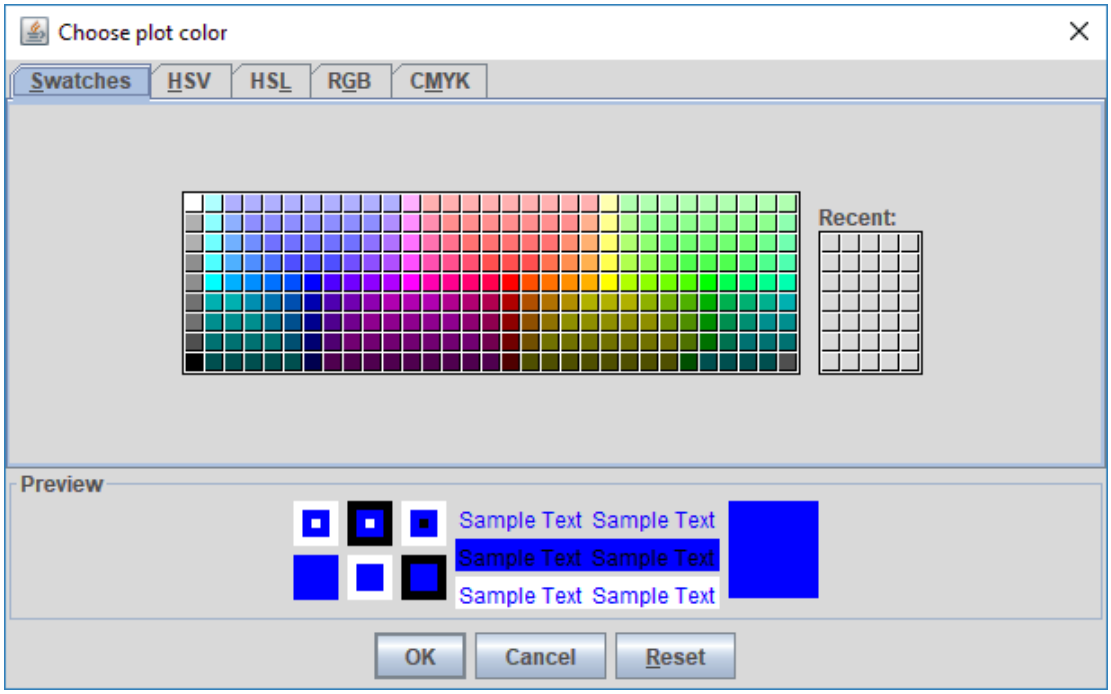


Рисунок 4.31. Вибір кольору.

	A	B	C	D	E	F
1	1,705891	1,764064	0,964498	3,02653		
2	1,357846	6,454832	0,803741	3,096344		
3	1,382273	15,17298	0,891662	9,928135		
4	1,66965	0,840292	0,841917	3,075019		
5	1,427666	2,979622	0,864471	0,870421		
6	1,427507	14,84033	0,745348	7,75937		

Рисунок 4.32. Приклад файлу з даними.

	A	B	C	D	E	F	G	H	I	J
1	Variable 0	Variable 1	Variable 2	Variable 3	Cluster 0 r	Cluster 1 r	Cluster 2 r	Cluster 3 r	Max membership	
2	1,705891	1,764064	0,964498	3,02653	0	1	0	0	1	
3	1,357846	6,454832	0,803741	3,096344	0	0	1	0	2	
4	1,382273	15,17298	0,891662	9,928135	1	0	0	0	0	
5	1,66965	0,840292	0,841917	3,075019	0,000148	0,791634	0,183731	0,024487	1	
6	1,427666	2,979622	0,864471	0,870421	3,61E-05	0,004784	0,993403	0,001777	2	
7	1,427507	14,84033	0,745348	7,75937	0,045947	0,000102	0,000212	0,953739	3	
8	1,47726	1,750947	0,777748	1,280094	7,03E-05	0,008462	0,984772	0,006695	2	

Рисунок 4.33. Приклад файлу з результатом кластеризації.

Висновки до розділу 4

У даному розділі дипломної роботи розглянуто проектування та розробку інформаційної системи, що здійснює розбиття множини об'єктів за допомогою нечіткого алгоритму k-середніх з обмеженою масою робочої області.

Був розроблений програмний додаток, що здійснює кластеризацію даних, що задані як матриця "об'єкт-властивості" за допомогою вказаного алгоритму, описано його можливості.

Усі задачі були виконані успішно і відповідають висуненим вимогам.

РОЗДІЛ 5
СТАРТАП-ПРОЕКТ

Програмна система дозволяє обирати кращий алгоритм для лінійної класифікації серед запропонованих, що дозволяє суттєво зекономити час при вирішенні подібних проблем.

Складові:

1) Підсистема зчитування даних.

2) Підсистема обробки даних.

3) Підсистема розрахунку алгоритму.

4) Підсистема побудови площини поділу виведення результату.

Можливості:

- Оцінка якості та швидкості різних алгоритмів класифікації для різних задач.

Бізнес-модель: орієнтація на клініки, медичні лабораторії, лікарні та дослідні інститути.

Цінний продукт.

1. Сукупність товарів-послуг (покращення товару-послуг).

Комплекс послуг: оцінка якості та швидкості різних алгоритмів класифікації для різних задач за рахунок використання різних алгоритмів класифікації та їх порівняльного аналізу.

2. Вирішує такі проблеми клініки.

- стратифікація пацієнтів.

3. Підтримка і сервіс програмної системи:

Додатковий сервіс надається шляхом підтримки системи розробником.

4. Продуктова лінійка (унікальність):

Унікальний алгоритм кластеризації.

Унікальність пропозиції (новизна, продуктивність, дизайн, ціна, економія на витратах, зниження ризику, доступність, зручність).

На ринку не існує аналогів у даної програмної системи.

Спочатку концентрація (фокусування) на одному сегменті (приватні клініки), потім розширення сегменту (державні заклади, міжнародний ринок).

Сегмент споживачів.

1. Ринок.

Ринок: середнє сегментування – надання послуг діагностичним клінікам. Спочатку послуга спрямована на приватні клініки.

В майбутньому планується розширення списку сегментів (міжнародний ринок).

2. Що об'єднує клініки - мета надати якісне медичне обслуговування.

3. Як вони звикли купувати, користуватися.

Купувати - додаток для комп'ютера.

Канали збуту.

1. Прямі – прямий продаж клінікам через сайт.

Фріміум – частина послуг (наприклад, інформаційна) безкоштовна, частина – платна

(або Shareware - 1 місяць безкоштовне надання послуг, далі – платно).

2. Функції.

Продажні (велике рішення) – в системі присутні всі модулі.

Надання рекомендацій в реальному часі – модуль вартістю 1000 грн.

Взаємодія із споживачами.

1. Залучення клінік, їх утримання.

Клініка купує систему.

2. Підтримка (пошта, телефон, особисто, форум).

Наприклад, на форумі можна повідомляти про внесення в систему вдосконалень (при цьому необхідні повторні продажі).

3. Залучення клієнтів завдяки науковим та медичним конференціям.

4. Повторний продаж системи завдяки реалізації додаткових модулів-послуг.

Дохід (монетизація).

1. За що і скільки готовий платити клієнт?

За систему, яка буде прац.ювати та підтримуватися розробником.

Ключові види діяльності.

1. Процес створення цінності.

Адміністрування, розробка системи (в тому числі і наукова діяльність), навчання користувачів системи, створення додаткових модулів (за потребою).

2. Виробництво товару-послуг, продаж.

3. Підтримка рішення.

Ключові ресурси.

1. Матеріальні (в т.ч. комп'ютерна техніка) - 3 персональні комп'ютери, принтер.

2. Інтелектуальні (в т.ч. патенти та ліцензії).

3. Людські:

- Бухгалтер веде всю фінансову діяльність фірми (нарахування і сплата податків, розподіл прибутку, розрахунок і видача зарплати).

- Старший програміст здійснює розробку програм, програмно-технічних засобів і контролює їх якісне виконання. Під його контролем працюють два фахівці в даній області, які й реалізують успішне виконання проекту.

- Маркетолог – за дослідження потреб, доцільності подальшого розширення системи,

- Генеральний директор займається кадрами, укладає договори на поставку продукції в організації та установи, відвідує виставки, конференції з обміну досвідом, відповідає за поставку обладнання у випадку його зносу,

технічного старіння. Забезпечує регулярну поставку сировини, проводить дослідження ринку, виконує розрахунки, пов'язані зі змінами в технології.

4. Фінансові - фінансування всіх членів команди (в т.ч., програмістів-розробників).

5. Час реалізації проекту.

Ключові партнери.

1. Забезпечення ресурсами (комп'ютерною технікою та обладнанням).

2. Оптимізація та економія в продажах.

3. Зниження ризиків і невизначеності.

4. Подальший розвиток проекту.

Витрати

Таблиця 5.1

Витрати на паливо й енергію на технологічні цілі

Назва	Кількість	Потужність обладнання, кВт	Корисний фонд часу, годин	Коефіцієнт за часом	Коефіцієнт за потужністю		Тарифи за одну кВт/год енергії, грн.
Комп'ютер	4	0,40	0,01	0,50	0,50	0,00	140,70
Всього						1,97	

Таблиця 5.2

Розрахунок собівартості одиниці продукції

Найменування статей калькуляції	Всього, грн.	Питома вага, %
Попутні комплектуючі вироби і напівфабрикати	166,4500	76,4470
Паливо й енергія на технологічні цілі	1,9698	0,9047
Основна заробітна плата	11,0000	5,0521
Додаткова заробітна плата	3,3000	1,5156
Відрахування на соціальне страхування	5,3768	2,4695

Продовж. табл. 5.1

Відшкодування зносу спеціальних інструментів	0,0019	0,0009
Витрати на утримання і експлуатацію обладнання	0,1670	0,0767
Загальновиробничі витрати	12,2837	5,6416
Виробнича собівартість	200,5492	92,1080
Адміністративні витрати	11,1670	5,1288
Інші витрати	2,0055	0,9211
Витрати на збут	4,0110	1,8422
Повна собівартість	217,7327	100,0000

Планується продаж 2000 ліцензій в місяць, вартість яких становить 435465,3726 грн.

Фінансовий план.

Витрати.

Таблиця 5.3

Витрати на місяць

Статті витрат	Сума, грн
1. Постійні витрати	44178,21
1.1 Орендна плата	7680,00
1.2 Заробітна плата працівникам	21000,00
1.3 Нарахування на заробітну плату	10836,00
1.4 Плата за телефон та інтернет	119,81
1.7 Витрати на електроенергію	3939,60
1.8 Амортизаційні відрахування	602,80
2. Змінні витрати (на перший місяць)	389587,70
2.1. Витрати на закупку матеріалів та техніки	383087,70
2.2. Витрати на рекламу	5000,00
2.3. Інші невраховані витрати	1500,00
Вартість устаткування	40000,00
Всього	473765,91

Постійні витрати:

- Орендна плата - 7680 грн/міс.
- Заробітна плата працівникам.
- Генеральний директор – 5000 грн.
- Бухгалтер – 4000 грн.
- Старший програміст – 4000 грн.
- Старший інженер – 4000 грн.
- Програміст – 3500 грн.
- Прибиральниця – 2000 грн.

Всього: 22500 грн.

Відрахування з заробітної плати:

- до пенсійного фонду – 35,2%;
- єдиний соціальний внесок – 16,4%;

Всього: 51,6%.

51,6% от 33000,00грн = 11 610 грн.

Амортизаційні відрахування.

- зношення обладнання становить 10% від балансової вартості обладнання на рік – 4000 грн., а в місяць – 334 грн;
- зношення споруди 3.5% від балансової вартості на рік – 3225,6 грн, а в місяць - 268, 8 грн.

Визначення місячної виручки.

ПЗ ми плануємо продавати в офісі за ціною 300 грн за 1 шт.

(300, 00 • 2000 = 600000 грн.).

Розрахунок точки беззбитковості.

Точка беззбитковості дозволить визначити, коли проект перестане бути збитковим (рис. 5.1).

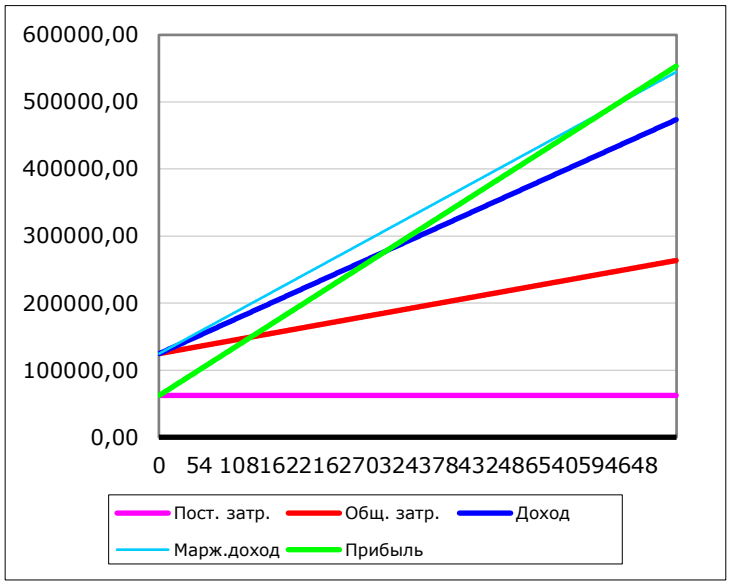


Рисунок 5.1. Точка беззбитковості

Вийшло, що величина точки беззбитковості дорівнює 613,98 штук, тобто необхідно випустити 614 ліцензій, і тільки після цього підприємство стане отримувати прибуток.

Окупність.

Рисунок 5.2. Період окупності

Дисконтований період окупності 2,6084 місяці.

Рисунок 5.3. Ставка дисконтування

IRR ≈ 32 % (ставка рентабельності за якої чиста приведена вартість 0).

Споживчі властивості товару.

Базові (очікувані) властивості:

- придбавши систему користувач очікує на якісний результат аналізу даних;

Основні (бажані) властивості

- зручність під час використання;
- точність результату;
- надійність у роботі.

Дослідження ринку.

Під час початкового періоду дослідження ринку потрібно відвідувати конференції з data mining, на яких розглядаються дані проблеми.

Дослідження конкурентного оточення.

Повний доступ до всіх ресурсів можна здійснити одразу після встановлення додатку на комп'ютер (тобто процедура реєстрації не потрібна). Користувачам видається повний доступ до інформації зі змогою редагування даних. Конкурентів у програмної системи немає.

План розвитку товару.

- надійність товару в споживанні: зведення помилок програми до мінімуму;

- ергономічні властивості: зручність експлуатації товару - зручний інтерфейс системи;
- естетичні властивості: здатність товару виражати свою соціокультурну значимість, ступінь корисності та досконаліть;
- безпека споживання: система є безпечною при використанні;
- як зміна характеристик продукту змінює споживчі властивості.

Додавання модулів до системи розширює споживчі властивості. Збільшення функцій (модулів) збільшує ціну товару.

Додаток використовують в комплексі із комп'ютером.

Управління ціною.

1. Формування ціни на продукт (витрати на розробку системи, організаційні заходи, рекламні заходи).

2. Формування системи лояльності: системи знижок, заохочень (для збільшення продажів, повторних продажів, партнерських продажів): робота із БД для покращення алгоритмів.

Висновки до розділу 5

Створено стартап-проект на основі магістерської дисертації. Розраховано його фінансовий плвн, точку беззбитковості. Створено план розвитку товару. Розраховано собівартість одиниці товару.

ВИСНОВКИ

1. У результаті виконання магістерської дисертації було проведено аналіз сучасних алгоритмів кластеризації. Проаналізовані алгоритми відносяться до різних типів та вирішують різні задачі. Наприклад алгоритми ієрархічної кластеризації часто використовуються в біології, для побудови філогенетичних дерев, а алгоритм FCM використовується для сегментації зображень.

2. Була здійснена розробка алгоритму нечітких k-середніх з обмеженою масою робочої області. До алгоритму було додано процедуру апріорної оцінки кількості кластерів на основі аналізу гістограм щільності розподілу змінних. Крім того до алгоритму додано розрахунок значення функції належності. Розрахунок функції належності дозволяє отримати додаткову інформацію про структуру кластерних утворень, а також здійснити поправки результату кластеризації k-середніх з обмеженою масою, що особливо важливо коли результат кластеризації отримується за один прохід.

3. За допомогою контекстної діаграми, моделі варіантів використання, діаграми станів, діаграми послідовності, діаграми кооперації, діаграми діяльності було проведено детальне проектування, що реалізує створені моделі. Програмний продукт було розроблено за допомогою мов програмування Java та Scala в середовищі IntelliJ IDEA. Він являє собою зручну програму для проведення кластеризації даних за допомогою розробленого алгоритму.

4. При перевірці роботи алгоритму на тестовому наборі даних був отриманий результат подібний до того, що був отриманий за допомогою метода Варда. Значення F_1 міри для розробленого алгоритму склало 0,92, а для методу Варда 0,9. При цьому розроблений алгоритм має нижчу обчислювальну вартість та використовує менше пам’яті, оскільки відсутня

необхідність у зберіганні та оновленні матриці відстаней. Крім того, оскільки розроблений алгоритм не є ієрархічним, тому для нього відсутня проблема знаходження місця зрізу дендрограми.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. B. Everitt Cluster Analysis / B. Everitt, S. Landau, M. Leese, D. Stahl – Wiley, 2010. – 346 p.

2. Паклин, Н. Алгоритмы кластеризации на службе Data Mining [Електронный ресурс] – Режим доступу до ресурсу: https://basegroup.ru/community/articles/datamining

3. Баранов, А. Data Mining. Теория и практика / А. Баранов, И. Брянцев, И. Жевлаков. БДЦ-пресс. - 2006. - С. 208.

4. Барсегян А. А. Технологии анализа данных: Data Mining, Visual Mining, Text Mining, OLAP. 2-ое издание Текст. / Барсегян А. А., Куприянов М. С., Степаненко В. В., Холод И. И. СПб.: БХВ-Петербург, 2007. 384 с.

5. Бериков, В.С. Современные тенденции в кластерном анализе / В.С.Бериков, Г.С. Лбов // Всероссийский конкурсный отбор обзорно-аналитических статей по приоритетному направлению «Информационно-телекоммуникационные системы». 2008. - 26 с.

6. Härdle, W. Applied Multivariate Statistical Analysis / W. Härdle, L. Simar -Springer-Verlag, Berlin-Heidelberg, 2007. 455 p.

7. Черезов, Д. С. Обзор основных методов классификации и кластеризации данных / Д. С. Черезов, Н. А. Тюкачев // Вестник ВГУ, серия: системный анализ и информационные технологии. — 2009. № 2. - С. 25-29.

8. Уманець В.С. Модифікований алгоритм k-середніх для функціонально зв'язних кластерів: диплом. Робота. Київ. «КПІ ім. Ігоря Сікорського», 2016

9. Смирнов Е. С. Таксономический анализ / Е. С. Смирнов. – М.: Издательство Московского университета, 1969. – 187 с.

10. Жамбю М. Иерархический кластер-анализ и соответствия. /М. Жамбю – М.: Финансы и статистика, 1988. – 344 с.

11. Ward, J. H., Jr. Hierarchical Grouping to Optimize an Objective Function / Ward, J. H., Jr.. // Journal of the American Statistical Association. – 1963. – №58. – С. 236–244.

12. R. Xu Clustering / R. Xu, D. Wunsch – Wiley, 2009. – 358 p.

13. Abonyi, Jânos. Cluster Analysis for Data Mining and System Identification / Jânos Abonyi, Balâzs Feil. — Springer, 2007. — P. 303.

14. Combining Sampling Techniquewith DBSCAN Algorithm for Clustering Large Spatial Databases / [H. Yunfa, F. Ye, W. Jin та ін.] // Knowledge Discovery and Data Mining. Current Issues and New Applications / [H. Yunfa, F. Ye, W. Jin та ін.]. – Berlin: Springer Berlin Heidelberg, 2000. –P. 169–172.

15. Density-Based Clusteringin Spatial Databases: The Algorithm GDBSCAN and Its Applications / J.Sander, M. Ester, H. Kriegel, X. Xu // Data Mining and Knowledge Discovery / J.Sander, M. Ester, H. Kriegel, X. Xu. – Berlin: Springer-Verlag, 1998. – P. 169–194.

16. Density-based clustering / [H. Kriegel, P. Kröger, J. Sander та ін.]. // Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. – 2011. – №3. – С. 231–240.

17. OPTICS: Ordering Points To Identify the Clustering Structure / [M. Ankerst, M. KrögerM. Breunig, H. SanderKriegel та ін.]. // ACM SIGMOD international conference on Management of data. – 2011. – С. 49–60.

18. Нейронные сети Кохонена [Електронний ресурс] – Режим доступу до ресурсу: http://neuronus.com/theory/961-nejronnye-seti-kokhonena.html.

19. Kohonen T. Self-Organizing Maps / Teuvo Kohonen., 2001. – 501 с.

20. В.А.Головко, под ред. проф. А.И.Галушкина Нейронные сети: обучение, организация и применение. – Москва:ИПРЖР, 2001

21. Борисов Е. Кластеризатор на основе нейронной сети Кохонена. [Електронний ресурс] / Евгений Борисов. – 2014. – Режим доступу до ресурсу: http://mechanoid.kiev.ua/neural-net-kohonen-clusterization.html.

22. Загоруйко Н. Г. Алгоритмы обнаружения эмпирических закономерностей / Николай Григорьевич Загоруйко., 1985. – 107 с.

23. Воронцов К. В. Лекции по алгоритмам кластеризации и многомерного шкалирования [Електронний ресурс] / К. В. Воронцов. – 2007. – Режим доступу до ресурсу: http://www.ccas.ru/voron/download/Clustering.pdf.

24. Демидова Л.А. Классификация объектов на основе мультимножеств и нечеткой кластеризации // Известия ТРТУ. Таганрог, 2006. - № 15 (70). - С. 72-79.

25. Рыбин, В.В. Основы теории нечетких множеств и нечеткой логики/ В.В. Рыбин. М.: МАИ, 2007. - 96 с.

26. S. Miyamoto Algorithms for Fuzzy Clustering: Methods in c-Means Clustering with Applications / S. Miyamoto, H.Ichihashi, K. Honda – Springler-Verlang Berlin Heidelberg, 2008. – 229 p.

27. Островский А.А. Вариант параллельного выполнения алгоритма FCM–кластеризации //V-я Международная научно-практическая конференция "Интегрированные модели и мягкие вычисления в искусственном интеллекте" (28-30 мая 2009 г., Коломна, Россия): Сборник научных трудов. В 2-т., М: Физматлит, 2009, Т.2, с.886-896

28. Синюк В.Г., Анищенко А.И. Нечёткое С-среднее и метод нечёткого роя частиц для решения проблем нечёткой кластеризации //XIII национальная конференция по искусственному интеллекту с международным участием КИИ-2012 (16-20 октября 2012 г., Белгород, Россия): Труды конференции. В 4-т., Белгород: Изд-во БГТУ, 2012, Т.4, с.27-34

29. Елизаров С. И. Адаптивные методы нечеткой кластеризации Текст. // Сб. докл.: Международная конференция по мягким вычислениям и измерениям SCM, 2007. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2007. - Т.2. - С.234-237.

30. What is Java [Електронний ресурс] – Режим доступу до ресурсу: https://java.com/ru/download/faq/whatis_java.xml.

31. Tour of Scala. Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://docs.scala-lang.org/tour/tour-of-scala.html.

32. Основи Swing. Використання JLabel [Електронний ресурс] – Режим доступу до ресурсу: https://schoolboyprog10.blogspot.ru/p/swing-gui-java.html.

33. F1-score [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/F1_score.

34. Apache POI [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Apache_POI.

35. Rousseeuw P.J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis // Journal of Computational and Applied Mathematics. – 1987. – 20, pp. 53-65.

36. E.R. Hruschka, L. Vendramin, and R.J. G.B. Campello. On the comparison of relative clustering validity criteria // 2009 SIAM International Conference on Data Mining (SDM 09)

37. Настенко Е.А. Закономірності самоорганізації та регуляції кровообігу людини : дис. ... д-ра биол. наук : 03.00.02 / Настенко Е.А. ; Нац. ин-т сердечно-сосудистой хирургии им. Н.Н. Амосова АМН Украины – К.,2008. – 36 с.

38. Настенко Є.А. Уманець В.С. Метод нечітких k-середніх з обмеженою масою робочої області формування кластерів довільної форми // Біомедична інженерія і технологія, 7 с.

39. Уманець В.С. Модифицированный алгоритм С-средних для функционально связных кластеров // Теорія і практика наукових знань (частина IV): матеріали II Міжнародної науково-практичної конференції м. Київ, 28-29 грудня 2017 року. – Київ.: МЦНД, 2017. С. 48-49.

40. Scott D.W. On optimal and data-based histograms // Biometrika. 1979. V.66. P. 605-610.

41. Іриси Фішера [Електронний ресурс] – Режим доступу до ресурсу: http://archive.ics.uci.edu/ml/datasets/Iris.

42. Kullback S. Letter to the Editor: The Kullback-Leibler distance // The American Statistician. – 1987. – 41(4): pp. 340–341.

43. Інформаційні технології та моделювання бізнес-процесів / [О. М. Томашевський, Г. Г. Цегелик, М. Б. Вітер та ін.]. – К.: Видавництво «Центр учбової літератури», 2012. – 296 с.

44.Нотация IDEF0 [Електронний ресурс] – Режим доступу до ресурсу: http://www.businessstudio.ru/wiki/docs/v4/doku.php/ru/csdesign/bpmodeling/idef0.

45. Диаграмма вариантов использования (use case diagram) [Електронний ресурс] – Режим доступу до ресурсу: http://www.info-system.ru/designing/methodology/uml/theory/use_case_diagram_theory.html.

46. Диаграмма состояний (statechart diagram) [Електронний ресурс] – Режим доступу до ресурсу: http://www.intuit.ru/studies/courses/1007/229/lecture/5954?page=4.

47. Диаграммы кооперации и их нотация [Електронний ресурс] – Режим доступу до ресурсу: http://www.intuit.ru/studies/courses/1007/229/lecture/5960?page=3.

48. Диаграмма последовательностей (sequence diagram) [Електронний ресурс] – Режим доступу до ресурсу: http://www.intuit.ru/studies/courses/1007/229/lecture/5954?page=3.

49. Диаграмма активностей [Електронний ресурс] – Режим доступу до ресурсу: http://www.intuit.ru/studies/courses/1007/229/lecture/5958.

50. Richet Y. JMathPlot: interactive 2D and 3D plots [Електронний ресурс] / Yann Richet – Режим доступу до ресурсу: https://github.com/yannrichet/jmathplot.

Додаток А

Лістинг коду

Clustering util:

```
package com.maleficus.scala

import com.maleficus.clustering.Cluster
import com.maleficus.utils.DataProcessingUtil

import scala.collection.JavaConverters.asScalaBuffer
import scala.collection.mutable.{ArrayBuffer, ListBuffer}
import scala.util.Random
import scala.util.control.Breaks._


object ClusteringFP{

  type Point = Array[Double]

  case class ClusteringResult(clusters: List[Cluster], clusterAndMembership:
Array[Array[Double]])

  def perform(dataOrig  : Array[Point],
                           capacity  : Int,
                           forget    : Int,
                           n         : Int,
                           method                 :(Array[Point],    Int)=>Array[Point]    =
initCentroidsPeripheral):ClusteringResult={

       if(dataOrig==null || capacity<0 || forget<0 || n<2)
               throw new IllegalArgumentException()
    var data = DataProcessingUtil.normalize(dataOrig)
    val                    center                    =                    {data.reduceLeft((a,
b)=>a.zip(b).map((pair)=>pair._1+pair._2)).map(_/data.length.toDouble)}
    //data = data.sortBy((point)=>distance(point, center)) //data arranged from center of mass
         val centroids = initCentroids(data, n, method)
         val clusters = for(i <- 0 until n) yield new Cluster(capacity, forget, i.toString)
    val result = new ArrayBuffer[Array[Double]]
         clusters.zip(centroids).foreach((pair)=>pair._1.initCentroid(pair._2))
    for(point<-data){
      val cluster = clusters.minBy((c)=>distance(point, c.getCentroid))
      cluster.addPoint(point)
      result+= Array(cluster.getTag.toDouble)
    }
    new ClusteringResult(clusters.toList, result.toArray)
  }

  private  def  initCentroids(data:  Array[Point],  n:  Int,  method:  (Array[Point],  Int)  =>
Array[Point]): Array[Point] = {
    method(data, n)
  }

  private def initCentroidsRandom(data: Array[Point], n: Int): Array[Point] = {
    Random.shuffle(data.toList).take(n).toArray
  }

  /**
    *  First create (point, distance) pairs from data array. Overall distance is the sum of the
    * distance to the center of mass and distances to all other centroids. Then the result is
being
    * checked for not being <i>Nothing</i>
    * @param data An array of points
    * @param n Number of clusters
    * @return An array of centroids as an array of arrays of double values
  */
  private def initCentroidsPeripheral(data: Array[Point], n: Int):Array[Point]={

       val result = new ListBuffer[Point]()
       val                    center                    =                    data.reduceLeft((a,
b)=>a.zip(b).map((pair)=>pair._1+pair._2)).map(_/data.length.toDouble)
       for(i<-0 until n){
              result+= {
        data.map((point)=>(point,      distance(point,      center)+result.foldLeft(0.0)((acc,
p)=>acc+distance(p, point))))
                .maxBy((pointAndDist)=>pointAndDist._2) match {
```

```
          case (p, _)=>p
          case _ =>throw new Exception("Exception while initializing centroids as peripheral
points.")
      }
       }
    }
       result.toArray
  }

  private def initCentroidsCenterOfMass(data: Array[Point], n: Int):Array[Point]={
    val                    center                    =                    data.reduceLeft((a,
b)=>a.zip(b).map((pair)=>pair._1+pair._2)).map(_/data.length.toDouble)
    data.sortBy((point)=>distance(point, center)).take(n)
  }

  private def initCentroidsNearZero(data: Array[Point], n: Int):Array[Point]={
    val zero = new Array[Double](data(0).length)
    data.sortBy((point)=>distance(point, zero)).take(n)
  }

  def          formTotalResult(pointAndCluster:          Array[Array[Double]],          fuzzy:
Array[Array[Double]]):Array[Array[Double]]={
    pointAndCluster.zip(fuzzy).map((pair)=>(pair._1++pair._2))
  }

  def       membership(data:       Array[Point],       fuzzyCoef:       Double,       clusters:
List[Cluster]):Array[Array[Double]]={
    val result = data.map(
    (point)         => clusters.map(
    (cluster)       => belongDegPoints(point, cluster, fuzzyCoef, clusters)).toArray)
    result.map((el) => el :+ el.zipWithIndex.maxBy((pair) => pair._1)._2.toDouble)
  }

  def      membershipTr(data:      Array[Point],      fuzzyCoef:      Double,      clusters:
List[Cluster]):Array[Array[Double]]={
    val result = data.map(
    (point)         => clusters.map(
    (cluster)       => belongDegTrace(point, cluster, fuzzyCoef, clusters)).toArray)
    result.map((el) => el :+ el.zipWithIndex.maxBy((pair) => pair._1)._2.toDouble)
  }

  private    def    belongDeg(p:    Point,    cluster:    Cluster,    fuzzy:Double,    clusters:
List[Cluster]):Double={
    val res = clusters.foldLeft(0.0)(
      (acc,     el)     =>     acc+Math.pow(distance(p,     cluster.getCentroid())/distance(p,
el.getCentroid()), 2.0/(fuzzy-1)))
    if((1/res).isNaN)
      1.0
    else
      1/res
  }

  private   def   belongDegPoints(p:   Point,   cluster:   Cluster,   fuzzy:Double,   clusters:
List[Cluster]):Double={
    val points = cluster.getPoints
    val res = clusters.foldLeft(0.0)(
      (acc,            el)=>acc+            Math.pow(asScalaBuffer(points).foldLeft(0.0)((sum,
point)=>sum+distance(p, point))/(points.size.toDouble-1)/
        (asScalaBuffer(el.getPoints).foldLeft(0.0)((sum,               point)=>sum+distance(p,
point))/el.getPoints.size.toDouble), 2.0/(fuzzy-1)))
    if((1/res).isNaN)
      1.0
    else
      1/res
  }

  private   def   belongDegTrace(p:   Point,   cluster:   Cluster,   fuzzy:Double,   clusters:
List[Cluster]):Double={
    val trace = cluster.getTrace
    val res = clusters.foldLeft(0.0)(
      (acc, el)=>acc+ Math.pow(asScalaBuffer(trace).map((point)=>distance(p, point)).min /
        (asScalaBuffer(el.getTrace).map((point)=>distance(p, point)).min), 2.0/(fuzzy-1)))
    if((1/res).isNaN)
      1.0
    else
      1/res
  }
```

```
  def silhouette(clusters: List[Cluster]):Array[Double]={
    for(cluster<-clusters)yield {
      val points = asScalaBuffer(cluster.getPoints)
      val combined = points.flatMap((p1)=>points.map((p2)=>(p1, p2)))
      val a = combined.foldLeft(0.0)((acc, el)=>acc+distance(el._1, el._2))/(combined.length-
1.0)
      val b = {
        for{cluster1<-clusters if cluster1!=cluster} yield {
          val combined = points.flatMap((p1)=>asScalaBuffer(cluster1.getPoints).map((p2)=>(p1,
p2)))
          combined.foldLeft(0.0)((acc,                              pair)=>acc+distance(pair._1,
pair._2)/combined.length.toDouble)
        }
      }.min
      (b-a)/Math.max(a, b)
    }
  }.toArray



  private def distance(a: Point,  b: Point): Double={
       a.zip(b).foldLeft(0.0)((acc, cort)=>acc+Math.pow(cort._1-cort._2, 2))
  }

  def estimate(dataNotNorm: Array[Point]):Int={
    val data = DataProcessingUtil.normalize(dataNotNorm)

    val dim = data.head.length
    val results = new Array[Int](dim)
    //val grouped = new mutable.HashMap[Double, Int]()

    for(i<-0 until dim){
      val sum = data.foldLeft(0.0)(  (acc, el) => acc+el(i))
      val  s  =  Math.sqrt(data.map(               (el)  =>  el(i)).foldLeft(0.0)((acc,  el)  =>
acc+1/(data.length-1.0)*Math.pow(el-sum/data.length, 2)))
      val step = 2.5*s*Math.pow(data.length, -1/3.0)
      val groupsCount = (1/step).toInt
      val grouped = data.groupBy(      (point) => (point(i)/step).toInt)
      grouped.foreach(                    (el) => print((el._1, el._2.size) + " "))
      println()

      for(group<-1 until groupsCount){
        breakable{
          if(!grouped.exists((keyval) => keyval._1==group-1)||
          !grouped.exists((keyval)    => keyval._1==group)||
          !grouped.exists((keyval)    => keyval._1==group+1))
            break
          val currGroup = grouped.get(group).getOrElse(new Array(0))
          val prevGroup = grouped.get(group-1).getOrElse(new Array(0))
          val nextGroup = grouped.get(group+1).getOrElse(new Array(0))

          if(prevGroup.size < currGroup.size &&
             nextGroup.size < currGroup.size){
            results(i)= results(i) + 1
          }
        }
      }

      results.foreach((el)=>print(el + " "))
      println()
    }
    results.max
  }

  def initRandom(): (Array[Point], Int)     => Array[Point] = initCentroidsRandom
  def initPeripheral(): (Array[Point], Int) => Array[Point] = initCentroidsPeripheral
  def initNearZero(): (Array[Point], Int)   => Array[Point] = initCentroidsNearZero
  def initNearCenter(): (Array[Point], Int) => Array[Point] = initCentroidsCenterOfMass
}
```

Excel util:

```
package com.maleficus.utils;

/**
 *
 * @author Vitalii Umanets
 */
```

```
import org.apache.commons.io.FileUtils;
import org.apache.poi.openxml4j.exceptions.InvalidFormatException;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Random;


public class ExcelDataLoader {

    /**
     * Loads data from the given file as an array of numeric values.
     * @param path The path of the *.xlsx file to read.
     * @param listNmbr Number of the list to read.
     * @param omitFirstRow If the first row has names then this flag should be set to true to
ignore it.
     * @throws IOException
     * @throws InvalidFormatException
     * @return Data loaded from the file as an array of
     */
    public static double[][] loadXLSXFile(String path, int listNmbr, boolean omitFirstRow)
throws IOException, InvalidFormatException{
        XSSFWorkbook wb = new XSSFWorkbook(new File(path));
        XSSFSheet sh = wb.getSheetAt(listNmbr);
        int shift = omitFirstRow? 1 : 0;
        int rowsNmbr = sh.getLastRowNum()+1-shift;
        double[][] data = new double[rowsNmbr][sh.getRow(shift).getPhysicalNumberOfCells()];
        for (int i=0;i<data.length;i++) {
            for (int j = 0; j < data[0].length; j++) {
                data[i][j] = sh.getRow(i).getCell(j).getNumericCellValue();
            }
        }
        return data;
    }

    public static void saveXLSXFile(String path, double[][] data, double[][] result)throws
IOException{

        XSSFWorkbook wb = new XSSFWorkbook();
        XSSFSheet sheet = wb.createSheet();
        for(int i=0;i<data.length;i++){
            XSSFRow row = sheet.createRow(i);
            int column = 0;
            for(double elem : data[i]){
                XSSFCell cell = row.createCell(column);
                column++;
                cell.setCellValue(elem);
            }
            //Empty cell to distance result from points
            row.createCell(column);
            column++;
            for(double elem : result[i]){
                XSSFCell cell = row.createCell(column);
                column++;
                cell.setCellValue(elem);
            }
        }
        try (FileOutputStream out = new FileOutputStream(path)) {
            wb.write(out);
        }
    }

    public static void saveXLSXFile(String path, int variables, double[][] data)throws
IOException{

        XSSFWorkbook wb = new XSSFWorkbook();
        XSSFSheet sheet = wb.createSheet();
        XSSFRow row = sheet.createRow(0);
        XSSFCell cell;
```

```
        for(int i=0;i<data[0].length;i++){
            cell = row.createCell(i);
            if(i<variables)
                cell.setCellValue("Variable "+ i);
            else if (i==variables)
                cell.setCellValue("Cluster");
            else
                cell.setCellValue("Membership for cluster "+ (i-variables-1));
        }
        for(int i=1;i<=data.length;i++){
            row = sheet.createRow(i);
            int column = 0;
            for(double elem : data[i-1]){
                cell = row.createCell(column);
                column++;
                cell.setCellValue(elem);
            }
        }
        try (FileOutputStream out = new FileOutputStream(path)) {
            wb.write(out);
        }
    }

    public static void saveXLSXExistingFile(String dataPath, double[][] data) throws
IOException, InvalidFormatException{
        File result = new File(dataPath.replace(".xlsx", " result.xlsx"));
        FileUtils.copyFile(new File(dataPath), result);
        FileInputStream inputStream = new FileInputStream(result);
        XSSFWorkbook wb = new XSSFWorkbook(inputStream);
        XSSFSheet sh = wb.getSheetAt(0);

        int columnsVar = sh.getRow(0).getLastCellNum();
        sh.shiftRows(0, sh.getLastRowNum()+1, 1, true, true);
        int rows = sh.getLastRowNum();
        Row row = sh.createRow(0);
        for(int i=0;i<columnsVar;i++){
            row.createCell(i).setCellValue("Variable " + i);
        }
        int columns = row.getLastCellNum();
        row.createCell(columns).setCellValue("Cluster");
        for(int i=1;i<data[0].length-1;i++){
            row.createCell(columns+i).setCellValue("Cluster " + (i-1)+" membership");
        }
        row.createCell(columns+data[0].length-1).setCellValue("Max membership");
        for(int i=0;i<rows;i++){
            row = sh.getRow(i+1);
            int startCol = row.getLastCellNum();
            for(int j= 0;j<data[i].length;j++){
                row.createCell(startCol+j).setCellValue(data[i][j]);
            }
        }
        inputStream.close();
        try (FileOutputStream out = new FileOutputStream(result)) {
            wb.write(out);
        }


    }

}
```

Java clustering adapter:

```
package com.maleficus.clustering;

import com.maleficus.utils.DataProcessingUtil;

import java.util.*;
import java.util.concurrent.atomic.AtomicInteger;
import com.maleficus.scala.ClusteringFP;


public class Clustering {

    private static final org.apache.log4j.Logger LOGGER =
org.apache.log4j.Logger.getLogger(Clustering.class);
    public static final double DEFAULT_RADIUS = 0.1;
    private final int forget;
```

```
    private final int capacity;
    private final int number;


    public enum InitMethod {RANDOM, NEAR_ZERO, NEAR_CENTER, PERIPHERAL}
    private InitMethod initMethod;
    private double[][] data;
    private List<Cluster> clusters;
    private int fuzzynessCoef;
    private double radius;

    /**
     * Math
     * @param clusters Number of clusters.
     * @param data
     * @param capacity
     * @param fuzzyness
     */
    public static Clustering init(int clusters, double[][] data, int capacity, int fuzzyness,
int forget){
        if(clusters<2)
            throw new IllegalArgumentException("Illegal number of clusters.");
        if(data.length<clusters)
            throw new IllegalArgumentException("Data has insufficient points.");
        if(capacity<1)
            throw new IllegalArgumentException("Illegal capacity value.");
        if(fuzzyness<1)
            throw new IllegalArgumentException("Illegal fuzzyness value.");
        if(forget<1 || forget>capacity)
            throw new IllegalArgumentException("Illegal number of points to forget.");
        return  new Clustering(clusters, data, capacity, fuzzyness, forget);
    }

    private Clustering(int numClusters, double[][] data, int capacity, int fuzzyness, int
forget) {
        this.data = data;
        this.clusters = new ArrayList<>();
        this.fuzzynessCoef = fuzzyness;
        this.radius = DEFAULT_RADIUS;
        this.initMethod = InitMethod.RANDOM;
        this.forget = forget;
        this.number = numClusters;
        this.capacity = capacity;
        //for(int i=1;i<=numClusters;i++)
        //    this.clusters.add(new Cluster(capacity, forget, String.valueOf(i)));
    }

    public InitMethod getInitMethod() {
        return initMethod;
    }

    public void setInitMethod(InitMethod initMethod) {
        this.initMethod = initMethod;
    }

    public ClusteringFP.ClusteringResult start(){
        //prepareData();
        //DataProcessingUtil.printData(data, 2);
        ClusteringFP.ClusteringResult clusters;
        switch (initMethod){
            case PERIPHERAL:
                clusters = ClusteringFP.perform(data, this.capacity, this.forget, this.number,
ClusteringFP.initPeripheral());
                break;
            case NEAR_ZERO:
                clusters = ClusteringFP.perform(data, this.capacity, this.forget, this.number,
ClusteringFP.initNearZero());
                break;
            case NEAR_CENTER:
                clusters = ClusteringFP.perform(data, this.capacity, this.forget, this.number,
ClusteringFP.initNearCenter());
                break;
            default:
                clusters = ClusteringFP.perform(data, this.capacity, this.forget, this.number,
ClusteringFP.initRandom());
                break;
        }
```

```
        double[][] dataToSave = ClusteringFP.formTotalResult(clusters.clusterAndMembership(),
                ClusteringFP.membershipTr(data, fuzzynessCoef, clusters.clusters()));
        scala.collection.Iterator<Cluster> iterator = clusters.clusters().iterator();
        for(int i=0;i<number;i++){
            Cluster clust = iterator.next();
            List<double[]> points = new ArrayList<>();
            for(int j=0;j<dataToSave.length;j++){
                if(i==(int)dataToSave[j][dataToSave[j].length-1])
                    points.add(data[j]);
            }
            clust.points = points;
        }
        DataProcessingUtil.printData(dataToSave, 2);
        clusters = new ClusteringFP.ClusteringResult(clusters.clusters(), dataToSave);
        return clusters;//assignPoints(result);
    }
}
```

Cluster class:

```
package com.maleficus.clustering;

import com.maleficus.utils.DataProcessingUtil;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class Cluster {

    //private final List<double[]> points;
    private double[] centroid;
    private final String tag;
    private final int MAX_CAPACITY;
    private double capacity;
    private int forget;
    private double[] sum;
    List<double[]> points;
    private List<double[]> trace;

    public Cluster(int  capacity, int forget, String tag) {
        //this.points = new ArrayList<>();
        MAX_CAPACITY = capacity;
        this.forget = forget;
        this.capacity = capacity;
        this.tag = tag;
        this.points = new ArrayList<>();
        this.trace = new ArrayList<>();
    }

    public String getTag() {
        return tag;
    }

    public synchronized void addPoint(double[] point){
        if(sum==null)
            sum = new double[point.length];
        if(capacity>0){
            capacity-=1;
            for(int i=0;i<point.length;++i){
                sum[i]+=point[i];
                centroid[i] = sum[i]/(MAX_CAPACITY-capacity);
            }
            points.add(point);
            trace.add(Arrays.copyOf(centroid, centroid.length));
        }else{
            for(int i=0;i<point.length;++i){
                sum[i]*=1-forget/(double)MAX_CAPACITY;
            }
            capacity+=forget;
            addPoint(point);
        }
    }

    public void initCentroid(double[] centroid){
        this.centroid = Arrays.copyOf(centroid, centroid.length);
        //this.sum = Arrays.copyOf(centroid, centroid.length);
```

```
        trace.add(this.centroid);
    }

    public double[] getCentroid() {
        return Arrays.copyOf(centroid, centroid.length);
    }

    public int getCapacity() {
        return MAX_CAPACITY;
    }

    /**
     * Returns list of points which belong to this cluster
     * @return <b>Immutable</b> list of <code>double[]</code>
     */
    public List<double[]> getPoints() {
        return Collections.unmodifiableList(points);
    }

    public List<double[]> getTrace() {
        return Collections.unmodifiableList(trace);
    }
}
```

UI controller:

```
package com.maleficus.ui.controller;

import com.maleficus.clustering.Cluster;
import com.maleficus.clustering.Clustering;
import com.maleficus.scala.ClusteringFP;
import com.maleficus.utils.ExcelDataLoader;
import javafx.application.Platform;
import javafx.embed.swing.SwingNode;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.Node;
import javafx.scene.control.Button;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import org.apache.poi.openxml4j.exceptions.InvalidFormatException;
import org.math.plot.Plot3DPanel;
import scala.collection.JavaConverters;

import javax.swing.*;
import java.awt.*;
import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.util.List;
import java.util.ResourceBundle;

public class MainForm implements Initializable{

    private static final org.apache.log4j.Logger LOGGER =
org.apache.log4j.Logger.getLogger(MainForm.class);

    @FXML private Button btnStartClustering;
    @FXML private Label lblPath;
    @FXML private Label lblNumPoints;
    @FXML private Label lblNumVariables;
    @FXML private TextField txtNumOfClusters;
    @FXML private TextField txtFuzzyness;
    @FXML private TextField txtClustCapacity;
    @FXML private TextField txtClustForget;
    @FXML private ChoiceBox<String> choiceInitMethod;
    @FXML private SwingNode plotNode;
    @FXML private Label lblResultQuality;
    private Plot3DPanel plot3d;
    private FileChooser fileChs;
    private String path;
    private double[][] data;

    @Override
```

```
    public void initialize(URL location, ResourceBundle resources) {
        fileChs = new FileChooser();
        fileChs.getExtensionFilters().add(
                new FileChooser.ExtensionFilter("Excel data sheet", "*.xlsx")
        );
        SwingUtilities.invokeLater(()->{
            plot3d = new Plot3DPanel();
            plot3d.setPreferredSize(new Dimension(600, 600));
            plotNode.setContent(plot3d);
        });

    }

    public void startClustering(){
        try{
            Clustering                              clusteringProcess                              =
Clustering.init(Integer.valueOf(txtNumOfClusters.getText()),
                    data,
                    Integer.valueOf(txtClustCapacity.getText()),
                    Integer.valueOf(txtFuzzyness.getText()),
                    Integer.valueOf(txtClustForget.getText()));
            String initMethod = choiceInitMethod.getValue().replace(' ', '_').toUpperCase();
            clusteringProcess.setInitMethod(Clustering.InitMethod.valueOf(initMethod));
            System.out.println(Clustering.InitMethod.valueOf(initMethod));
            ClusteringFP.ClusteringResult result = clusteringProcess.start();
            double[] silhouette = ClusteringFP.silhouette(result.clusters());
            List<Cluster> clusters = JavaConverters.seqAsJavaList(result.clusters());
            ExcelDataLoader.saveXLSXExistingFile(path, result.clusterAndMembership());
            plot3d.removeAllPlots();
            for(Cluster cluster :clusters){
                List<double[]> points = cluster.getPoints();
                double[][] scatterData = new double[3][points.size()];
                for(int j=0;j<scatterData[0].length;j++){
                    scatterData[0][j] = points.get(j)[0];
                    scatterData[1][j] = points.get(j)[1];
                    scatterData[2][j] = (points.get(j).length>2)?points.get(j)[2]:0;
                }
                plot3d.addScatterPlot("Cluster "+cluster.getTag(), scatterData);

                List<double[]> trace = cluster.getTrace();
                double[][] traceData = new double[3][trace.size()];
                for(int j=0;j<traceData[0].length;j++){
                    traceData[0][j] = trace.get(j)[0];
                    traceData[1][j] = trace.get(j)[1];
                    traceData[2][j] = (trace.get(j).length>2)?trace.get(j)[2]:0;
                }
                //plot3d.addLinePlot("Cluster " +cluster.getTag()+" trace", traceData);
            }
            StringBuilder resultSil = new StringBuilder();
            resultSil.append("Silhouette Index\n\n");
            for(int i=0;i<silhouette.length;i++){
                resultSil.append("Cluster " + i + " : " + silhouette[i] + "\n");
            }
            lblResultQuality.setText(resultSil.toString());
//            for(int i=0;i<points.length;i++){
//                double[][] scatterData = new double[3][points[i].size()];
//                for(int j=0;j<scatterData[0].length;j++){
//                    scatterData[0][j] = points[i].get(j)[0];
//                    scatterData[1][j] = points[i].get(j)[1];
//                    scatterData[2][j] = points[i].get(j)[2];
//                }
//                plot3d.addScatterPlot("Cluster "+i, scatterData);
//            }

//            scatterChart.setData(FXCollections.observableArrayList());
//            for(int i=0;i<points.length;i++){
//                XYChart.Series<Number, Number> series = new XYChart.Series<>();
//                series.setName("Cluster "+(i+1));
//                points[i].stream().forEach((point)->{
//                    series.getData().add(new XYChart.Data<>(point[0], point[1]));
//                });
//                scatterChart.getData().add(series);
//            }

        }catch(Exception ex){
            LOGGER.error(ex);
            ex.printStackTrace();
        }
```

```
    }

    public void openFile(){
        File file = fileChs.showOpenDialog(null);
        if(file.exists()){
            this.path = file.getAbsolutePath();
        }
        try {
            data = ExcelDataLoader.loadXLSXFile(path, 0, false);
        } catch (IOException e) {
            LOGGER.error(e);
        } catch (InvalidFormatException e) {
            LOGGER.error(e);
        }
        txtNumOfClusters.setText(""+ClusteringFP.estimate(data));
        System.out.println(path);
    }

    public void closeApp(ActionEvent e){
        final Node node = (Node)e.getSource();
        final Stage stage = (Stage) node.getScene().getWindow();
        stage.close();
        Platform.exit();
    }

}
```

Main:

```
package com.maleficus;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class MainApp extends Application{


    public static void main(String[] args){
        launch(args);
    }

    @Override
    public void start(Stage stage) throws Exception {
        String fxmlFile = "/forms/MainForm.fxml";
        FXMLLoader loader = new FXMLLoader();
        Parent root = loader.load(getClass().getResourceAsStream(fxmlFile));
        stage.setTitle("Clustering");
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.setOnCloseRequest(event -> Platform.exit());
        Platform.setImplicitExit(true);
        stage.show();

    }

}
```